AALBORG UNIVERSITET
STUDENTERRAPPORT

GENERATIVE DESIGN SPACE UDFORSKNING

EN GENETISK ALGORITME TILGANG TIL PERFORMANCE BASED DESIGN

Byggeri og Anlæg
4. Semester Bygningsinformatik
Sylvester Knudsen

School of Engineering and Science
Thomas Manns Vej 23
DK - 9220 Aalborg Øst
Tlf. 99 40 93 09
www.ses.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Titel:	Generative Design Space Udforskning
Semester:	4. Semester
Semester tema:	Kandidatspeciale
Projektperiode:	1. september 2017 – 10. januar 2018
ECTS:	30
Vejleder:	Kjeld Svidt

Sylvester Holm Knudsen

SYNOPSIS:

Denne rapport er udarbejdet som kandidat speciale på Aalborg Universitets Cand.Tech Bygningsinformatik.

Specialet arbejder med en teknisk løsning, der forsørger at skabe et bedre beslutningsgrundlag i den tidlige designfase, ved at bruge Performance Based Design.

Videns grundlaget er skabt på baggrund af videnskabelige artikler, samt en virksomheds specifik case.

Digital publicering:	**PDF**
Antal sider:	77
Anslag:	**152.263**
Bilag:	**1 stk.**

FORORD

Dette projekt er udarbejdet som kandidatspeciale på Cand.Tech i Bygningsinformatik uddannelsen ved Aalborg Universitet, i perioden 1. september 2017 – 10. januar 2018.

Igennem rapporten undersøges det hvordan Performance Based Design, ved hjælp af Generative design og optimerings algoritmer, kan understøtte en mere data driven design proces. Rapporten indledes med et problembehandlingsafsnit, hvor problemstillingen beskrives fra to perspektiver, det større brancheproblem beskrevet ved hjælp af den videnskabelige litteratur, og et mere firmaspecifikt problem der beskrives ved hjælp af en case. Efterfølgende gives en introduktion til de elementer der bruges i projektets løsning. Sidst er der udarbejdet et løsningsforslag, der gennem de beskrevet elementer, giver en forslag til hvordan problemstillingen kan løses.

Rapporten ligger stor vægt på den tekniske udarbejdelse af løsningen. Der er derfor stor fokus på at omsætte den teoretiske viden, givet i løsnings grundlag afsnittet, til en praktisk fungerende løsning.

I forbindelse med udarbejdelsen af dette projekt, skal der gives et stort tak til MT Højgaard, der har været en stor hjælp under problembehandlingsdelen. Ligeledes har de under hele forløbet stillet nødvendigt software og testcases til rådighed. Ydermere har de stille specialist viden til rådighed, til vejledning og evaluering af løsningen.

Derudover skal der siges tak til vejleder og lektor på Aalborg Universitet Kjeld Svidt, for igennem hele projektforløbet at have bidraget med god rådgivning og relevant sparring.

ABSTRACT

This project is made as the final thesis at Aalborg University, MSc. Building Informatics, in the period 1. September 2017 – 10. January 2018.

The project looks at how Performance Based Design can secure at better decision process in the design phase, using Generative desing and a optimization algorithm.

The thesis starts by outlaying the problem of the project, based on the literature study the problem as looked at as an industry problem. Performance simulations in the early design phase is beneficial for the performance of the building for the entire lifespan, although this is the case researchers has found that most of the available Building Performances Simulation (BPS) tools, is lacking the ability to be used in the early design phase. Most of today's BPS tools, focus on optimizing a given design based on one criteria thereby a sub optimization process occurs and optimization changes becomes hard to do.

The thesis then moves on to describe elements that can help to do better BPS in the early design phase. Generative design is a process that helps explore a large design space for optimal solutions, when combining a parametric model and an optimizing search algorithm, a design solution can be evolved based on different objectives. By using these elements, the digital tools becomes more of partner to the designer, a partner that helps make the best decisions. By using the computer to search through the design space, a much larger set of alternatives can be covered, giving the designer a better decision basis.

Lastly the thesis describes the developed solution. The aim of the solution is to develop a workflow that by combining elements like Generative design and Performance Based Design, gives a much better foundation for BPS in the early design phase. The showcased workflow describes one way of better integrating BPS in design, and doing it in an optimized and automated way.

INDHOLDSFORTEGNELSE

I. PROBLEMBEHANDLING

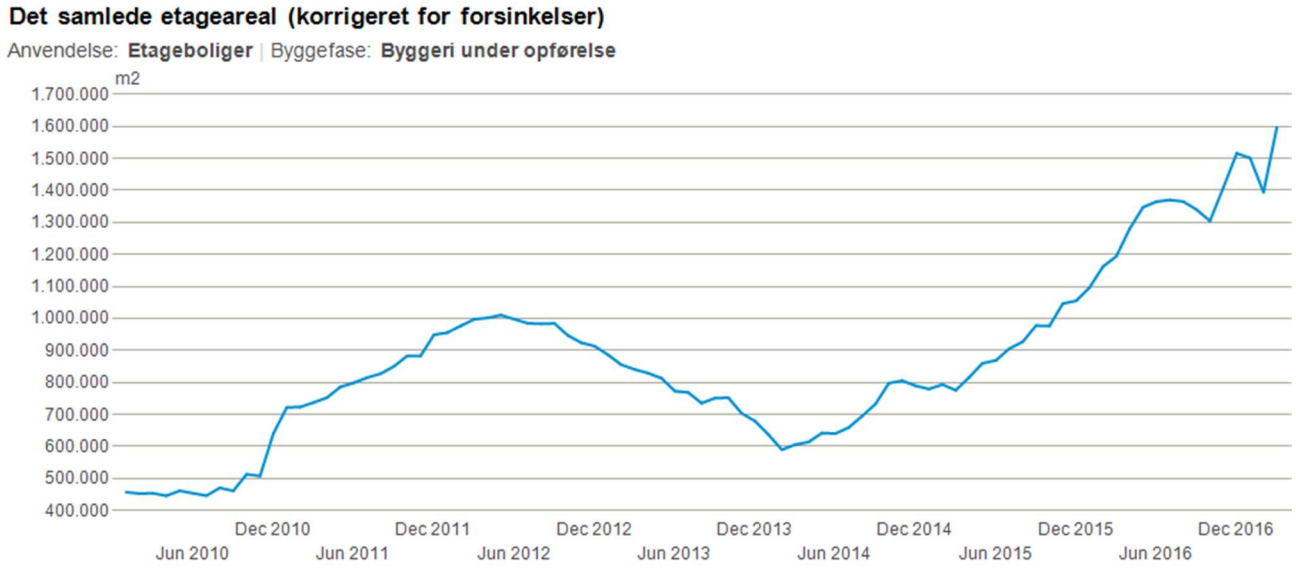
1 INDLEDNING

I mange større byer, både internationalt og nationalt, opleves der et stigende behov for flere boliger. Især i tiderne omkring studiestart fyldes medierne med historier om, hvordan de større studiebyer ikke har boliger nok til alle de nye tilflyttere. Ligeledes ses det, at der fra befolkningens side er et større ønske om at bo i større byer, hvilket ligner en trend der vil forsætte i fremtiden (Rothenborg 2017).

Alene i Københavns kommune ses en massiv befolkningsvækst, hvilket gør manglen på boliger et mere og mere presserende problem.

> *"Københavns Kommunes seneste befolkningsprognose fra marts 2017 viser, at de nu 600.000 københavnere vil blive til 710.000 i 2030. Og væksten forventes at fortsætte næsten uændret, så der er tæt på 800.000 i 2040."* (Rothenborg 2017).

I takt med stigning i ønsket om at bo i større byer og den generelle stigning i befolkningsantallet, udvikles og opføres der flere boliger. Nedenstående graf viser udviklingen af det samlede etageareal af etageboliger under opførelse fra start 2010 til slut 2016. Som grafen tydeligt viser, har der de seneste år været en markant stigning på hele 250%.



Figur 1-1 Graf over udviklingen af det samlede etageareal af etageboliger under opførelse (Kilde: Danmarks statistik)

Den store fremgang i udvikling af nye etagebygninger skaber samtidigt en større interesse for at opføre markante og utraditionelle bygninger. Disse etageboliger opføres i højere grad som høje huse, altså bygninger hvor antallet af etager er over seks. Typisk vælges de høje huse for at profilere virksomheder eller for at fungere som varetegn for byen. Det ses også, at de høje huse er nødvendige for at udnytte by arealerne bedst muligt (Teknik & Miljø 2006).

1.1 SIMULATION BASED DECISIONS

To byggeprojekter er sjældent ens, dog deler alle projekter nogle fælles karakteristiske træk, som til sammen danner byggeprocessen. Som med de fleste typer projekter

inkluderer et byggeprojekt generelt planlægning, udvikling og implementering af projektet. Bygningsdesigns udfordres ofte ved de strenge krav, der i dag stilles til nye bygninger, bl.a. ved energi, omkostninger og den generelle performance af bygningen (Østergård et al. 2017). Derudover er der stort fokus på, hvordan nye bygningers design påvirker de eksisterende omgivelser, såsom skyggepåvirkninger og udsyn til interessepunkter.

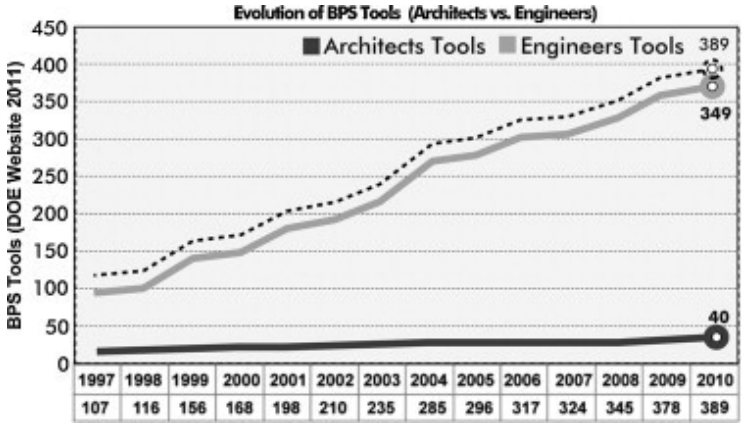
Byggeprojekter kompliceres yderligere, da der ofte er mange interessenter involveret med hver deres dagsorden og ansvarsområde. Alle parter påvirker projektet og dets omkostninger og har stor indflydelse på den endelige økonomi og performance af produktet. De tidlige faser af projektet er afgørende for det endelige produkts succesfaktor, og de valg, der træffes her, vil have betydning for mange af de valg, der træffes i de efterfølgende faser. De tidlige faser, hvor bygningsdesignets overordnede form udvikles, påvirkes af en lang række designparametre, der til sammen skaber et enormt *design space* (Østergård et al. 2017). Dette design space kan ses som et rum, hvor alle mulige designløsninger er repræsenteret ud fra en sammensætning af alle designparametrenes mulige værdier. Dermed kan der allerede ved forholdsvis få parametre være flere millioner forskellige designløsninger, et tal som bliver større jo flere parametre der kobles på. Derfor er det ganske enkelt umuligt at analysere alle kombinationer af designs, hvilket i mange tilfælde heller ikke er nødvendigt, da en række af designløsningerne hurtigt kan filtreres fra.

Det amerikanske forsknings- og rådgivningsfirma Gartner, som er specialister inden for teknologiforskning, har i samarbejde med den danske entreprenørvirksomhed MT Højgaard sammensat en omfattende rapport omkring potentialerne ved brugen af BIM (Philip & Philip 2014). I denne undersøgelse pointerer de, at der i gennemsnit kun analyseres på 2,7 alternativer i forbindelse med byggeprojekter. Set i sammenligning med hvor mange forskellige alternativer der er i det føromtalte design space, er det bemærkelsesværdigt få alternativer. Det konkluderes ligeledes i rapporten, at der med simple metoder til at analysere flere alternativer vil blive skabt grundlag for bedre og mere innovative bygninger.

> *"With simple ways to analyze multiple scenarios the final design can be more innovative, better for the environment and cheaper to build – all at the same time"* (Philip & Philip 2014)

Der findes i dag en lang række værktøjer, der hjælper med simuleringer af bygningsdesigns, hvilket bevidner om et stor fokus på udviklingen af disse værktøjer (se Figur 1-2). På trods af denne udvikling inden for simuleringsværktøjer er værktøjerne typisk fokuseret på at simulere allerede designede alternativer efter beslutningsprocessen i de tidlige faser (Østergård et al. 2015). Designprocessen af et byggeprojekt er traditionelt meget sekventiel opdelt, hvor designerne typisk udfører en række aktiviteter efter hinanden. Typisk ses det, at der udarbejdes et designforslag, som derefter kontrolleres mod de mål og krav, der er sat for byggeriet. Hvis kontrollen af designet ikke lever op til målene, må designeren gå tilbage i processen og rette de ting, som han mener kan ændre på kontrolresultatet. Denne proces forsætter iterativt, til der findes et design, der overholder de krav og mål, der er sat (Attia et al. 2012). De seneste år ses det i høj grad, at krav og performancemål til byggerier bliver strengere og strengere. Studier viser ligeledes, at 20% af de trufne beslutninger i de tidlige designfaser efterfølgende påvirker 80% af alle designbeslutninger (Bogenstätter 2000). Derfor er det ekstremt vigtigt, at de beslutninger, der tages i de tidlige faser, tages ud fra et velanalyseret grundlag. Det betyder også, at designere i de tidlige faser er

tvunget til at udvide deres ansvarsområde ud over funktion og æstetik og i højere grad integrere flere analyser i deres tidlige arbejde.

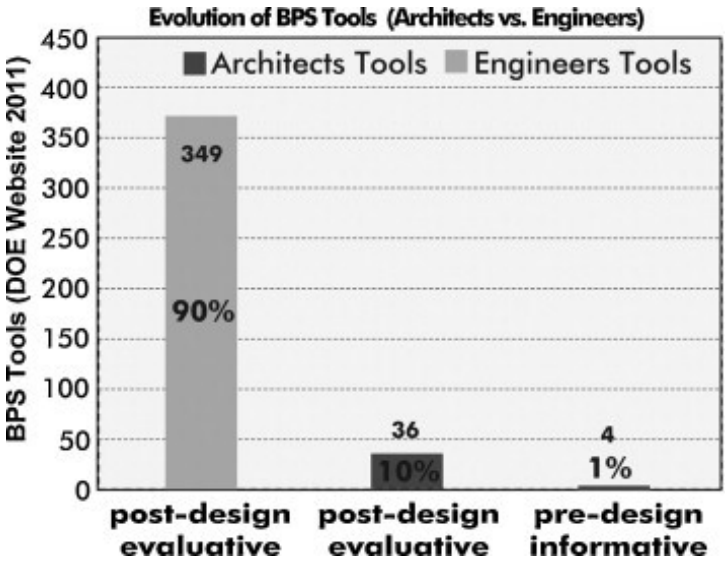


Figur 1-2 Udviklingen af Building Performance Simulation værktøjer (Attia et al. 2012)

Ofte vil et byggeprojekts designere i de tidlige faser ikke være dem, der sidder med den dybdegående viden omkring simulering og beregning af byggeriets performance. Designere i de tidlige faser er specialister inden for æstetik og udnyttelse af byggeriets funktion, og der vil derfor være en naturlig barriere, når de skal integrere flere performanceanalyser i deres arbejde. For at overkomme disse barrierer udvikles der som tidligere nævnt en lang række Building Performance Simulation (BPS) værktøjer. De fleste af disse simuleringsværktøjer er dog ikke i stand til tilstrækkeligt at give feedback om designet i de helt tidlige faser. Flere undersøgelser viser, at de nuværende værktøjer er utilstrækkelige, brugerfjendtlige og for ufuldstændige til at blive brugt af designere i de tidlige faser (Attia et al. 2009).

Der opstår hurtigt en barriere mellem designere i de tidlige faser og brugen af BPS værktøjer. Dette er på trods af, at forskning viser, at brugen af disse værktøjer i de tidlige faser er gavnlig for bygningens ydeevne resten af dens levetid. De fleste BPS værktøjer fokuserer på høj nøjagtighed og præcise simuleringer, hvilket kræver et højt informationsniveau. Ofte er det informationsniveau, der arbejdes med i den tidlige designfase meget begrænset, hvilket gør det svært at inkorporere BPS værktøjer. For at overkomme denne barriere, argumenterer Augenbroe (Augenbroe 2002) for en række punkter, som værktøjerne skal understøtte. Iblandt disse punkter er behovet for hurtigt at kunne evaluere design alternativer med feedback på performanceparametre. Derudover pointeres det også, at værktøjerne skal være mere intelligente, altså være i stand til assistere designeren i beslutningsprocessen. Det er derfor vigtigt, at den feedback, værktøjet giver, er intelligent og nem at kommunikere til ikke-eksperter inden for bygningsperformanceanalyser, et punkt der ligeledes bakkes op af Attita et al. (Attia et al. 2012). Tværfagligheden i værktøjerne er et andet vigtigt punkt i Augenbroes argumentation, det vil altså sige, at det skal være muligt at inkorporere flere analyser i simuleringsværktøjet. Tværfagligheden i analyserne er især vigtige, da ændringer på baggrund af én analyse kan have konsekvenser for en anden bygningsperformance analyse (Østergård et al. 2017).

Udvalget af BPS værktøjer er som tidligere beskrevet meget stort og udvikles i højt tempo. Building Energy Software Tools[1] er et godt sted at få et overblik over, hvilke BPS værktøjer der findes, og hvad deres fokus er. Undersøgelser udført af Attia et al. (Attia et al. 2012) understøtter yderligere det manglende fokus på designere i de tidlige faser, når det gælder BPS værktøjer. Deres undersøgelse viser, at ud af de 392 oplistede værktøjer var kun 40 af dem målrettet designere i de tidlige faser (se Figur 1-3).



Figur 1-3 opdeling af BPS værktøjer i før design og efter design (Attia et al. 2012)

Meget interessant viser undersøgelsen, at ud af de 40 værktøjer målrettet designere var kun fire af dem tiltænkt som et værktøj til at give informativ feedback under selve designprocessen. På trods af undersøgelsen efterhånden er seks år gammel, vurderes det overordnede billede stadig at være sigende for den nuværende situation.

På trods af det øgede fokus inden for området og den generelle hurtige udvikling inden for teknologi, viser litteraturen at BPS værktøjer som hjælp til designbeslutninger stadig er mangelfuld. Litteraturen viser dog samtidig et øget fokus på netop denne problemstilling med flere eksempler på værktøjer, der udvikles med fokus på de tidlige beslutningsfaser. Et godt eksempel på dette er værktøjet udviklet af Østergaard et al. (Østergård et al. 2017). Værktøjet er ud fra en geometrisk model i stand til at simulere konsekvensen af flere tusinde designparametre.

[1] https://www.buildingenergysoftwaretools.com/software-listing - liste over tilgængelige BPS software, tidligere administreret af US Dept. of Energy

2 CASE

Følgende afsnit introducerer den case, som projektet bygges op omkring. Casens primære del omhandler et udviklingsprojekt for MT Højgaards VDC afdeling. Afsnittet vil kort forklare udviklingsprojektet samt beskrive det projekt, som er udleveret til at teste en løsning på.

2.1 MT HØJGAARD A/S

MT Højgaard (MTH) er en af nordens største og førende entreprenørvirksomheder med bygge- og anlægsprojekter i både ind- og udland. Virksomhedens historie går 100 år tilbage og er en sammenlægning af de to entreprenørvirksomheder Højgaard & Schultz og Monberg & Thorsen (MT Højgaard 2017).

Figur 2-1 MT Højgaards vision

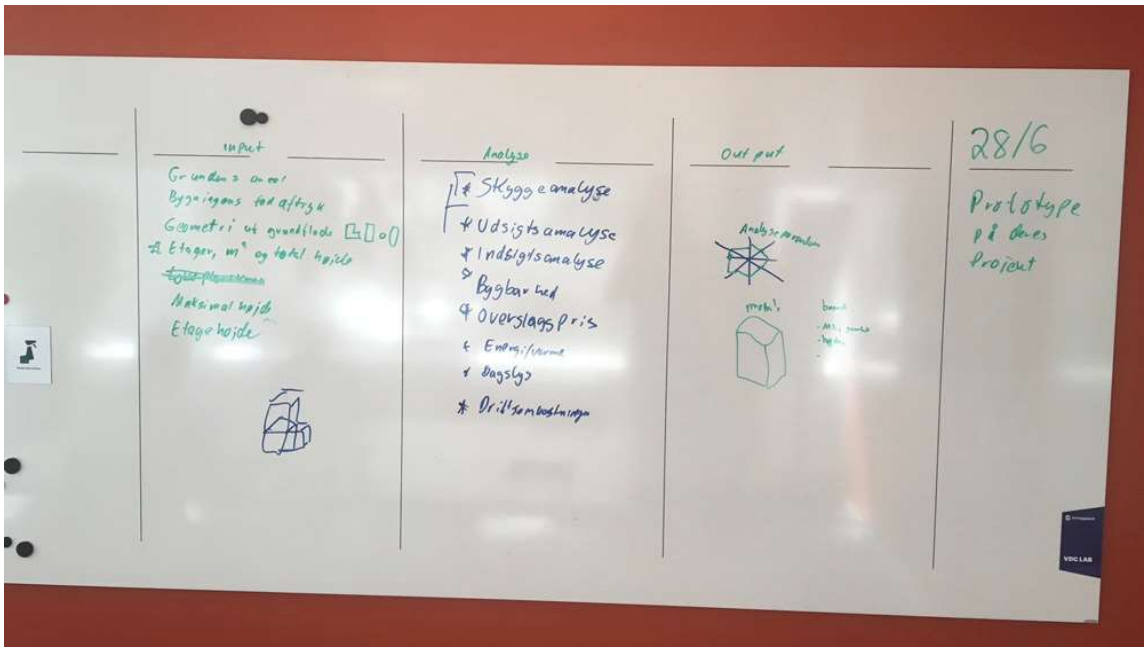
MTH's overordnede vision er at blive den mest produktivitetsfremmende koncern i bygge- og anlægsbranchen (Figur 2-1). Denne vision er opsplittet i nogle krav og mål, som skal medføre, at den overordnede vision opnås. Et af målene i visionen er at være best in class i Virtual Design and Construction (VDC). Dette mål viser, at MTH tror på udviklingen af digitale workflows i byggeriet som værende en af måderne at opnå bedre produktivitet på. De seneste år har MTH investeret massivt i udviklingen af BIM og VDC, hvilket ses tydeligt i VDC divisionen som i dag tæller over 50 mand. For at være best in class er det nødvendigt at være med fremme i udviklingen.

2.2 UDVIKLINGSPROJEKT

I forbindelse med MTH's udvikling inden for VDC, er computational design i høj grad blevet et emne, der udforskes i virksomheden. Computational design gør det muligt at udforske større design spaces, tage valg på et bedre datagrundlag og automatisere flere arbejdsgange. Dette åbner op for mulighederne for at optimere design processen både i analysedelen men også i forhold til at automatisere gentagende arbejder. På denne baggrund har virksomheden ønsket at udforske mulighederne for at genere et design på baggrund af en række analyser. Målet er altså at vende den traditionelle proces, hvor et design analyseres på baggrund af en fastlagt geometrisk form, til en proces hvor et design skabes ud fra, hvor godt det performer på en række parametre.

Rammerne for projektet er i høj grad frie, der er altså ingen umiddelbare retningslinjer for, hvordan løsningen udføres eller hvilke parametre der bruges i den. På et indledende møde med en VDC afdelingsleder, manager og specialist, er der lavet en brainstorm hvor løse tanker om indhold af løsningen er diskuteret (Figur 2-2). Selve

ideen er opstået grundet en række eksempler på udviklingsprojekter, hvor processen har været påvirket af ineffektivitet, fordi projektets designs for ofte ikke lever op til de opsatte krav. Der er derfor lagt op til en generisk løsning, der ved nye projekter kan tilpasses, så hvert projekts parametre kan måles og analyseres.

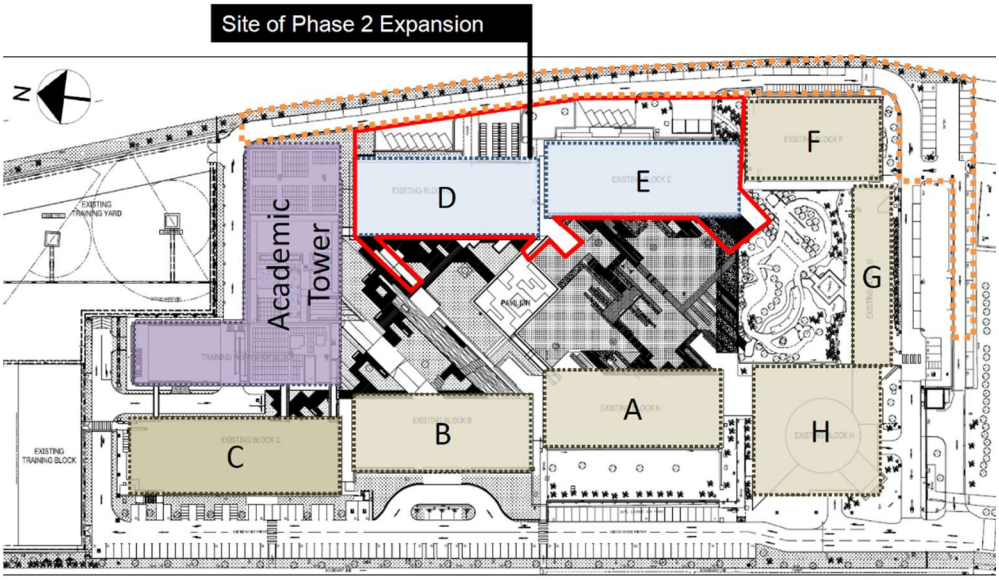


Figur 2-2 Resultat af det indledende brainstormmøde

Til udvikling af løsningen er der udleveret et projekt som kan bruges til at lave forsøg på. Projektet bruges som grundlag for løsningen, hvor dets forskellige parametre og krav bruges til at udvikle en prototype.

2.3 TESTPROJEKT

Projektet BCA Academy er en ny universitetsbygning, der skal opføres i Singapore. Den nye bygning skal placeres på det eksisterende campus og erstatter bygningerne D og E på Figur 2-3.



Figur 2-3 BCA Academy Masterplan

Bygning D og E erstattes med nye bygninger, der giver mere plads, hvor den ene bygning erstattes med en 7 etagers bygning på maks. 7.400 m², mens den anden bygges i 20 etager med maks. 20.000 m². Udover de opsatte krav til m² er der fra bygherre et ønske om at minimere skyggeforhold fra de nye bygninger på den eksisterende bygning A, da denne bygning har solceller monteret på taget.

I forbindelse med udviklingen af dette projekts løsning, fokuseres der udelukkende på at lave et design til bygning E.

3 PROBLEMFORMULERING

På baggrund af projektets case samt den mere generelle problembeskrivelse med afsæt i den videnskabelige litteratur, er den endelige problemformulering for projektet udarbejdet. Denne problemformulering vil danne grundlag for det resterende af denne rapport. Problemformuleringen tager udgangspunkt i det opstillede udviklingsprojekt fra MT Højgaard, men på baggrund af litteraturstudiet vurderes problemet som værende mere generelt for hele branchen.

Hovedproblemstillingen er formuleret som følgende:

Hvordan kan performance based design understøtte en mere datadriven beslutningsproces i den tidlige designfase?

På baggrund af denne hovedproblemstilling indledes rapporten med et redegørende afsnit, hvor de forskellige elementer, der udgør løsningen, introduceres for at give læseren et kendskab til de forskellige dele, der udgør den endelige løsning. Efterfølgende vil teorien bag disse elementer omsættes til en praktisk løsning, der udnytter de beskrevne elementer.

Til projektets hovedproblemstilling er der opstillet tre underspørgsmål, der skal hjælpe med at finde frem til den endelige løsning.

- Hvordan kan man udforske flere alternativer i et design space, samt optimere metode til at søge igennem et design space?
- Hvordan kan designproblemer optimeres ud fra flere parametre på samme tid?
- Hvordan kan åbne standarder bruges til at skabe et workflow, der kan måle sig med tilsvarende lukkede software?

Hovedproblemet og dets underspørgsmål behandles ud fra en naturvidenskabelig tilgang, hvor målet med projektet er at udvikle et teknisk workflow, der understøtter problemformuleringen.

4 AFGRÆNSNING

Udgangspunktet for en veldokumenteret rapport er en målrettet afgrænsning. Projektets afgrænsning er udført gennem 6 trin, hvis formål er at finde frem til kernen af det egentlige problem.

Afgrænsning		
1.	Øge datagrundlaget i den tidlige designbeslutningsproces	**Første ide**
2.	Kobling mellem form og analyseværktøj	**Afgrænsning**
3.	Perfomanceanalyseresultater der har indflydelse på design	**Yderligere afgrænsning**
4.	Designoptimering på baggrund af flere analyser	**Yderligere afgrænsning**
5.	Projektet er udarbejdet i et samarbejde med MT Højgaard og udvikles til deres behov. Det vurderes dog at være relevant i et større brancheperspektiv i forhold til at vurdere flere alternativer i designprocessen.	**Målgruppe**
6.	Muligheden for at koble flere analyser sammen med formgenerering i et workflow, der automatisk hjælper designere med at udvælge optimale designs.	**Fokusering**

Projektet tager udgangspunkt i det beskrevne udviklingsprojekt fra MT Højgaard, hvor der ønskes en metode til bedre at kunne integrere analyser med designformgivning. Nuværende software og processer udnytter ikke performancesimuleringer ordentligt i den tidlige designfase, men fokuserer mere på at analysere allerede formgivne designs. Dette resulterer i, at performanceanalyser og- simuleringer i højere grad bruges til at suboptimere designs frem for at være en drivfaktor i udviklingen af designet.

Ovenstående kan resultere i at designforslag skal igennem mange iterationer, der hver især fokuserer på at suboptimere på baggrund af enkelte analyser. Projektets fokus er at integrere form og analyseprocessen, samtidig med at flere analyser integreres i processen, så optimering af et design sker som en helhed fremfor som en suboptimering.

Overordnet set er projektet afgrænset sådan, at der udelukkende fokuseres på at udvikle en teknisk løsning. Den tekniske løsning udvikles på baggrund af det udleverede testprojekt samt testes ud fra dette projekts krav. Det er ikke målet, at den udviklede løsning skal kunne kopieres direkte til andre projekter, men løsningen skal ses som et workflow, hvor flere elementer kan bruges mellem projekterne, mens andre elementer skal udvikles specielt til et givent projekt.

5 METODE

Den metodiske tilgang er primært baseret på konkret udvikling og forsøg med bagvedliggende litteraturstudier. Nærværende projekt benytter en metode, hvor undersøgelser bygger på udvikling og test af en konkret løsning. Menneskelige og kulturelle faktorer i implementeringen af løsningen, er i dette projekt nedprioriteret for at fokusere på den tekniske udvikling.

5.1 DATAINDSAMLING

To primære metoder bruges til at indsamle det data, der skal bruges til at understøtte projektets problem og løsning, henholdsvis et litteraturstudie og en virksomhedscase.

5.1.1 LITTERATURSTUDIE

Litteraturstudie har fungeret som den primære kilde til at underbygge problemstillingen og den udviklede løsning. Et indledende studie er lavet, hvor projekter med lignende formål som dette er gennemgået. Ud fra det indledende litteraturstudie er et løsningsdomæne valgt, som der efterfølgende er foretaget et gennemgribende litteraturstudie på. Den primære kilde til litteratur har været videnskabelige artikler, abstracts fra konference har fungeret som supplerende kilder til de videnskabelige artikler, og derudover har enkelte bøger ligeledes været taget i brug. De videnskabelige artikler suppleret med konferenceabstracts har sikret et informationsgrundlag, der er up to date med nyeste teknologier.

5.1.2 CASE

Udover litteraturstudiet er der indhentet data fra den primære case, udviklingsprojektet udleveret af MT Højgaard, som består i et ønske om bedre at kunne benytte data til at genere et optimalt design. Derudover er der udleveret et specifikt projekt, der bruges som grundlag for test af løsningen. Selve problembeskrivelsen og konkretisering af udviklingsprojektet er udført på en workshop med relevante parter fra MT Højgaards VDC afdeling samt undertegnede.

5.2 LØSNINGSUDVIKLING

Projektets primære del er udvikling af en teknisk løsning, der afhjælper den primære problemformulering og dens underspørgsmål. Udviklingen af løsningen sker på baggrund af de beskrevne teknologier i rapportens hovedafsnit samt ønsker fra casen. Der er foretaget en afgrænsning af teknologier, der benyttes i løsningen. Denne afgrænsning er taget i det indledende litteraturstudie og er styrende for udviklingen af løsningen. Der findes mange måder og teknologier, der kan understøtte problemfeltet i dette projekt, men det ligger uden for projektets grænser at afdække alle disse, og derfor er der foretaget en litteraturstudiemæssig afgræsning af løsningsspektret til hvilke teknologier, der vil blive brugt i den endelige løsning.

I forbindelse med udviklingen af løsningen fokuseres der primært på at udvikle et teknisk workflow. Selve implementeringen af løsningen er derfor i dette projekt prioriteret lavere.

5.3 LØSNINGSTEST

Som opfølgning på den udviklede løsning foretages der til sidst en test og evaluering af workflowet. Testen sker ved at afprøve forskellige måde at køre workflowet på. Evalueringen sker ved hjælp af et specialistpanel, som præsenteres for løsningen hvorefter den diskuteres.

II. LØSNINGS GRUNDLAG

6 PARAMETRISK DESIGN

Følgende afsnit vil give læseren et indblik i, hvad parametrisk design er, hvorfor det bruges og hvilke udfordringer, der ligger bag. Parametrisk design er en central del af projektets løsning, ligesom parametrisk design er grundlaget for de resterende elementer der bruges i løsningen.

6.1 HVAD ER PARAMETRISK DESIGN

Ordet parametrisk stammer fra matematikkens verden, og hvornår man begyndte at bruge udtrykket i forbindelse med design, er der forskellige bud på. Flere argumenterer dog for, at *parametrisk design*-udtrykket stammer tilbage fra 1940'erne, hvor arkitekten Luigi Moretti, brugte udtrykket til at beskrive hvordan et stadion kunne designes på baggrund af 19 parametre (Davis 2013). I dag er parametrisk modellering en kendt metode i byggebranchen. Flere argumenterer for, at alt design er parametrisk design, hvilket bunder i tanken om, at den arkitektoniske proces altid har fungeret på en parametrisk måde. Andre har ligeledes stillet spørgsmål om, hvad ikke-parametrisk design er.

Parametriske modeller forbindes ofte med hurtige og nemme ændringer; ændres et parameter, ændres modellen på baggrund af den nye parameterværdi. Derfor beskrives de parametriske modeller ofte som værende meget fleksible og gode til at håndtere ændringer i designløsninger. Varemærket for parametriske modeller er derfor blevet, at ”de er nemme at ændre”.

Flere forfattere har identificeret to forskellige processer, der udgør en parametrisk model; brugen af modellen og skabelsen af den. Daniel Davis (Davis 2013) præsenterer i sin ph.d. afhandling begrebet *tooling* i forsøget på at beskrive, hvad en parametrisk model er. Tooling skal forstås som den proces, der gennemgås i forbindelse med at skabe den parametriske model til brug i designfasen. Igennem mange forskellige kilder viser Davis, hvordan disse to processer, brugen og skabelsen af den parametriske model, er adskilt fra hinanden og måske endda foregår i forskellige organisationer eller afdelinger. Robert Aish (Aish 2011) opdeler f.eks. handlingen i at skabe et værktøj og det at designe ved hjælp af et værktøj, idet han udtaler:

> *"Software developers do not design buildings. Their role is to design the tools that other creative designers, architects and engineers use to design buildings."*

Begrebet tooling dækker i denne sammenhæng over processen ved at skabe den parametriske model, som i høj grad kan sammenlignes med at udvikle et nyt designværktøj. Fokus på ikke at adskille selve skabelsen af den parametrisk model fra brugen af den bliver af flere anerkendt som værende vigtigt for udviklingen af dette område. Mark Burry mener bl.a., at man efterhånden ser en udvikling inden for brugen af værktøjer, hvor der før har været fokus på at være eksperter inden for brugen af et værktøj til i højere grad at være i stand til at udvikle vores egne værktøjer.

> *”we are moving rapidly from an era of being aspiring expert users to one of being adept digital toolmakers”* (Burry 2011)

Definitionen af en parametrisk model har som selve den parametrisk model mange forskellige inputs, der kan definere betydningen. Flere definitioner af en parametrisk model er givet i ovenstående, og ud fra disse synspunkter kan parametri være alt

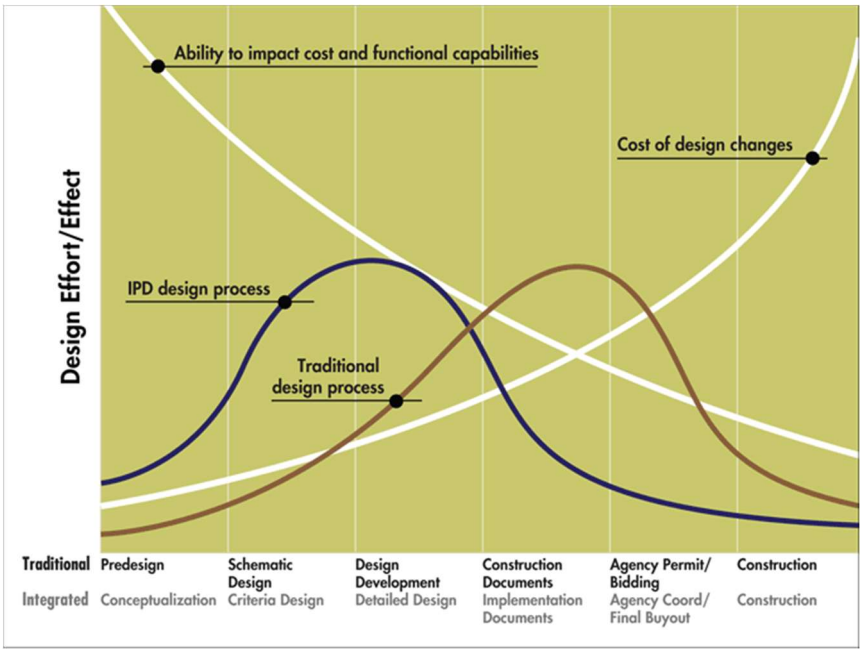
design, kun de designs der ændres eller *tooling*. En anden populær definition af parametriske modeller er, at en parametrisk model beskriver en fleksibel model. Fælles for disse definitioner er, at der lægges vægt på, hvad modellen gør, fremfor hvordan den er udviklet. Kigger vi tilbage på den matematiske verden, beskrives parametrisk som et sæt af ligninger, der definerer et output på baggrund af en eller flere variabler.

> *"a parametric equation is defined as a set of equations that express a set of quantities as explicit functions of a number of independent variables, known as 'parameters'"* (Weisstein 2003)

Den matematiske definition kan raffineres til brugen af parametriske modeller i byggebranchen ved, at outputtet af ligningen i de fleste tilfælde vil være geometri. En parametrisk model kan altså beskrives som et sæt ligninger, der gennem en række inputparametre beskriver en geometrisk model (Davis 2013).

6.2 HVORFOR PARAMETRISK DESIGN

Fokus på at optimere arbejdsprocesser har længe været et af de største emner i byggebranchen. Flere undersøgelser har vist, at jo længere et projekt er fremme, jo vanskeligere og dyrere er det at indføre nye ændringer (se Figur 6-1). De beslutninger, der træffes i starte af projektet, vil typisk have indflydelse på mange af de beslutninger der træffes i det videre projektforløb. Det vil altså sige, jo mere detaljeret et projekt er blevet, jo sværere er det at få ændringer gennemført.



Figur 6-1 MacLeamy kurven

Flere initiativer forsøger at gøre op med den mere traditionelle faseopdelte arbejdsgang, som er vist i Figur 6-1 ved den røde streg, heriblandt Building Information Modeling (BIM) og Integrated Project Delivery (IPD). Begge tiltag forsøger at gøre det nemmere at samarbejde og involvere de rigtige kompetencer på det rigtige tidspunkt, så flere beslutninger kan tages tidligt i projektet og derved holde ændringsomkostningerne nede. IPD og BIM forsøger begge at opmuntre designerne til at handle, når omkostningerne ved forandring er lave, og deres evne til at foretage forandringerne er høje. Problemet med at tvinge designbeslutninger til de tidlige faser

er, at projektet derved hurtigt bliver mere udviklet, hvilket ifølge MacLeamy's kurve (Figur 6-1) øger omkostningerne på ændringerne (Davis 2013).

Parametrisk modellering er en anden metode, der forsøger at sænke omkostningerne forbundet med design ændringer. Samuel Geisberg, grundlægger af Parametric Technology Corporation, indikerede i 1993, at man fremfor for at flytte alle designbeslutninger til de helt tidlige faser i stedet skulle forsøge at reducere omkostningerne ved designændringer (Davis 2013).

Fleksibilitet er et vigtigt aspekt af parametriske modeller. Ved at opretholde en høj grad af fleksibilitet i modellerne, muliggøres ændringer i designet. Netop muligheden for at fortage ændringer i designet, især i forbindelse med arkitektur, er ekstremt vigtigt for at sikre det optimale produkt. Mange faktorer kan nødvendiggøre designændringer, mange af dem er umulige at forudse og derfor planlægge, hvilket sætter store krav til fleksibiliteten af den parametriske model. Især faktorer udenfor designerens ansvarsområder er svære at forudse, politiske aspekter eller ændringer i markedspriser kan have stor indflydelse på bygningens design. Andre ændringer sker naturligt i et projektforløb, simpelthen fordi der opbygges en viden i projektet, som fører til ændringer. I de fleste tilfælde bliver mange problemstillinger samt deres løsninger kun kendte gennem iteration, udforskning og refleksion (Cross 2006). Set i lyset af disse uundgåelige ændringer, gør fleksibiliteten af en parametrisk model det til et tiltalende designmedium for designere (Davis 2013).

6.3 UDFORDRINGER MED PARAMETRISKE MODELLER

På trods af fleksibiliteten i parametriske modeller, som ifølge flere kilder skulle gøre designændringer nemmere at håndtere, ses det alligevel, at ændringshåndtering ikke er helt så simpelt alligevel. Andre studier viser, at designeren i mange tilfælde, er nødt til at omprogrammere hele den parametriske model for at inkorporere en ændring.

> *"Once you think you have a working parametric model you may still find you haven't programmed a parameter of the geometry in a way that is adjustable to a designer's future request. A designer might say I want to move and twist this wall, but you did not foresee that move and there is no parameter to accommodate the change. It then unravels your program. Many times you will have to start all over again."* (Smith 2007)

Rick Smith, pioner inden for parametrisk modellering, oplister i et whitepaper fem områder, der udfordrer en parametrisk models fleksibilitet. De fem problematiske områder bygger på erfaringer opnået gennem flere års arbejde med parametriske modeller (Smith 2007).

1. Parametriske modeller kræver høj grad planlægning
2. Behovet for fleksibilitet kan være svært at forudse
3. Store ændringer kan gøre modellen ubrugelig og kræve omprogrammering af hele modellen
4. Ændringer kan være svære at visualisere
5. Parametriske modeller er svære at genbruge og dele

Rick Smith påpeger lighederne mellem det at lave en parametrisk model og at udvikle software. Ligesom i softwareudvikling kræver en parametrisk model, at man på et konceptuelt plan har udtænkt, hvad det er, man i sidste ende skal modellere. Derefter kan den reelle programmering ske efterfulgt af omfattende test af modellen for at finde situationer, der vil få modellen til at fejle. I forbindelse med testning af modellen, er

der risiko for, at modellen bliver ”overbegrænset”, hvilket i sidste ende går udover fleksibiliteten.

På trods af omfattende test og planlægning kan det være ekstremt svært at forudsige enhver lille designændring, der måtte opstå i løbet af et projekt. Dette leder tilbage til ovenstående citat, som siger, at uforudsigelige ændringer i værste tilfælde kan føre til en fuldstændig omprogrammering af modellen.

Geometrien i en parametrisk model styres af forskellige inputparametre, som på kryds og tværs er forbundne, hvilket betyder, at ændringer et sted kan føre til ændringer et andet sted. Ændringerne kan derfor være svære at se, når de udføres, hvilket kan føre til uhensigtsmæssige ændringer, der i værste tilfælde først opdages, når byggeriet opføres.

Ovenstående hænger i høj grad sammen med Rick Smith’s femte punkt, der siger, at parametriske modeller er svære at dele, fordi det kræver stort kendskab til opbygningen af modellen at kunne navigere og manipulere den. Udover kendskab til selve opbygningen af modellen og dens logiske regler er der også store krav til, at brugerne er bekendte med det software, modellen er opbygget i. Den viden, der ligger bag opbygningen, er ofte meget svær at videregive, og på den måde bliver udvikleren af den originale model ofte en form for ejer af modellen. Kompleksiteten af en parametrisk model bliver hurtig meget omfattende, hvilket også gør, at det er meget svært for andre end den oprindelige udvikler at arbejde med modellen.

På baggrund af Smith’s whitepaper viser det sig især, at parametriske modeller, der anvendes i praksis, ofte udfordres af det, de især påstår at imødekomme, nemlig ændringer.

7 GENERATIVT DESIGN

Som beskrevet i projekts indledning er et bygningsdesigns *design space* enormt og indeholder mange designvarianter, mange designs der aldrig bliver afprøvet i designprocessen. Foregående afsnit gav en introduktion til, hvad parametriske modeller er, hvordan de kan bruges samt nogle af de udfordringer, der er forbundet med dem. Parametriske modeller baseres på algoritmer, der styrer et designs geometri på baggrund af forskellige inputparametre, hvilket giver beregningsmæssig kontrol over designmodellen. Evnen til at ændre et design på baggrund af designkriterier og krav, gør især parametriske modeller ideelle til at udforske det uudnyttede design space. Især når man kigger på *performancebaseret design,* er den parametriske kontrol af geometrien værdifuld, da man kan integrere performanceanalyser i en samlet designsyntese.

John Gero (Gero 1994) nævner i forordet til bogen *"Formal Design Methods for CAD"*, to keypoints i udviklingen af CAD:

1. The representation and production of the geometry
2. The representation and use of knowledge to support or carry the synthesis of designs

Det første punkt lægger sig I høj grad op ad *off-the-shelf* CAD software, hvis primære opgave er at øge effektiviteten og automatisere standard designopgaver. Det andet punkt lægger mere op til ideen om at samle og integrere viden i en pakke, der med hjælp fra computerkræfter kan assistere designere i at udforske nye ideer af et design. Generative designsystemer gør det muligt at udforske komplekse sammensætninger af et design gennem simple operationer med modellens parametre. Parametrisk design kan agere både som en generativ - og analyserende metode i forbindelse med udforskning af et design space.

7.1 PARAMETRISKE MODELLER SOM GENERATIVE SYSTEMER

Ordet design har dobbelt betydning, det kan være design som en aktivitet, altså det at designe noget, eller design som en artefakt, selve outputtet af designaktiviteten. Forskellen mellem disse to betydninger er central i generative designsystemer. Et generativt designsystem repræsenterer specifikationen af artefaktets designprocedure og har dermed fokus på dannelse frem for form, hvilket indikerer et skifte fra modellering af et designet objekt til modellering af et designs logik (Leach 2009). Generative designsystemer kræver specifikationerne for princippet af designartefaktet, hvor disse specifikationer kan være regler, parametriske afhængigheder, begrænsninger osv. På denne måde åbnes op for et design space, hvor forskellige variationer og alternative designs kan udforskes. Et generativt system skal imidlertid ikke ses som en erstatning af den menneskelige designer, mere som et redskab der assisterer designeren ved at påtage sig nogle af de tunge beregningsopgaver (Dino 2012). Med et generativt designsystem forsøger man altså at indkapsle intelligens til at udføre nogle af designopgaverne i et system, der langt hurtigere end noget menneske kan udarbejde løsninger på baggrund af den indprogrammerede intelligens. Processen i et generativt designsystem består af fire elementer: startbetingelser og parametre (inputs), en generativ mekanisme (regler, algoritmer osv.), generering af forskellige variationer (outputs) og udvælgelse af den mest optimale variant (Dino 2012).

En af metoderne til at skabe et generativt system er grundet dets algoritmiske fundament og evne til at udforske et design space parametrisk design. Algoritmisk tænkning og - design har stor betydning for generativt design. En algoritme er en beregningsmetode, der adresserer et problem i et begrænset antal trin. Algoritmer kan

betragtes som forlængelser af den menneskelige hjerne og kan lette springet til at udforske områder med uforudsigeligt potentiale (Terzidis 2011). Typisk forbindes algoritmer med programmeringsverdenen, hvor forskellige algoritmer bruges til at styre software og systemer. Det ses dog i højre grad at byggebranchen begynder at tage programmering til sig, og det er ikke usandsynligt at firmaer har ansatte, der kan begå sig i lettere scripting og simpel programmering. Øget produktivitet i forbindelse med at udforske designalternativer og større kontrol over designs ved at frigøre sig fra begrænsningerne i standard black-box[2] software er blandt fordelene ved at benytte scripting i byggebranchen (Burry 2011). Ved at udnytte scripting i et 3D-modelleringsmiljø opnås en større frihed og kontrol, der giver designeren mulighed for at udvide funktionalitet samt evaluere og agere på forskellige forhold. Som udgangspunkt er algoritmisk og parametrisk design det samme, hvor man kan argumentere for at parametrisk design er en underkategori til algoritmisk design. Rent beregningsmæssigt er der ikke forskel på de to, idet algoritmer opererer på parametre, mens parametriske systemers kerne er selve algoritmen (Dino 2012). Helt lavpraktisk kan man definere parametrisk design som en proces, hvor man fokuserer på hvilke parametre, der skal bruges og lader en algoritme bearbejde disse parametre til et endeligt design. Hvor algoritmisk design har et bestemt sæt parametre, og i modsætningen til parametrisk design ændres algoritmen for at skabe det endelige design.

7.2 PARAMETRISK DESIGN SPACE UDFORSKNING

Parametrisk design gør det muligt både at løse veldefinerede problemer med et klart mål men også mere komplekse problemer, der kan have flere brugbare løsninger. Et typisk byggeprojekt tilhører kategorien med mere komplekse problemer, ofte er der mere end én rigtig løsning. At designe et projekt er en proces uden en definitivt løsning, der er mere rigtig end andre løsninger, og ændringshåndtering er derfor vigtigt i designprocessen. Projektets designer har behov for kontinuerligt at kunne definere, omdefinere og ændre projektet for at nå det ønskede resultat. I et designprojekt ses det ofte, at selve problemet ikke er klart defineret af bygherren (Cross 2001). Som beskrevet flere gange tidligere er evnen til at udforske flere alternativer kritisk i forhold til at finde det optimale design. Akin (Akin 2001) beskriver forskellen mellem en ekspertdesigner og en nybegynder som evnen til at vurdere flere alternativer.

Konventionelt CAD softwares fokus er ofte den endelige designform, og typisk arbejder denne type software med et enkelt designalternativ. Parametrisk design er ikke afhængigt af et enkelt design, men åbner i stedet op for det store design space af mulige løsninger. Dermed får designeren mulighed for at udforske en række designmuligheder i løbet af projektet, gennemgå tidligere alternativer og hele tiden forbedre designet.

I forbindelse med at udforske design spaces kan en parametrisk model fungere som grundlag for automatisk generering af alternative designvariationer. En ændring i et inputparameter vil udløse en ændring i den geometriske form, samtidig med den overordnede struktur og de opsatte regelsæt for modellen overholdes. Den

[2] I videnskab, databehandling og ingeniørverden er en black-box et objekt, system eller en enhed, der gennem dens input uden kendskab til dets interne arbejde, generer et output. Næsten alting kan blive omtalt som en sort boks: en transistor, en algoritme eller den menneskelige hjerne.

parametriske model giver altså mulighed for at udforske langt flere alternativer af et design space.

8 PERFOMANCE BASED DESIGN

Følgende afsnit giver en introduktion til konceptet Performance Based Design, som er et nøgleemne i projektets løsning. Der gives ligeledes en introduktion til en metode, som projektets løsning har draget inspiration fra.

Traditionelt set har designere designet en bygning eller et område efter deres intuition og tidligere erfaringer. Når designet er på plads, overtager en ingeniør projektet, hvorefter diverse performanceanalyser foretages, og ud fra disse scenarier vælges den bedste løsning (Moradi 2014). Problemet med dette workflow er, at resultater af analyser som *building performance* sjældent ændrer på selve bygningens form, og ændringen ofte kræver mange ressourcer at indføre (Anton & T□nase 2016). Building performance skal ses i kontekst til det enkelte projekt og kan have en meget bred betydning. Performance dækker over mange felter som økonomi, arealplanlægning, samfundsindvirkning, kultur, teknologi eller miljøpåvirkninger. I nærværende projekt ses building performance som et udtryk for bygningens påvirkning på omgivelserne og miljøet.

Selvom der i dag er en lang række værktøjer, der kan fortage performance measurement analyser af et bygningsdesign, har disse målinger ikke nok indflydelse på selve bygningens form. Værktøjerne bruges til at dokumentere og lave mindre justeringer fremfor at blive brugt som styringsredskab, der assisterer designeren i den arkitektoniske formgivningsproces (Anton & T□nase 2016). Nye databehandlingsværktøjer til performance measurments kan i samarbejde med parametriske modeller bruges som bindeled mellem analyser og den formgivende proces i de tidlige designfaser (Oxman 2008).

Ved at kombinere performance med formgenerering gøres performance til endnu et parameter i den algoritme, der skaber et design, hvilket sikrer, at flere kriterier har indflydelse i den tidlige designfase. Gennem en parametrisk model hvor den formgivende proces omdannes til en algoritme, kan performanceværktøjer integreres til at give feedback på designet og skabe en generativ formskabelsesproces (Anton & T□nase 2016). Performance skal dog ikke ses som den altoverskyggende faktor for et bygningsdesign men som endnu et element i den større formgivningsalgoritme. Ved alene at kigge på selve performancedelen vil bygningslandskabet blive præget af meget funktionalistiske designs uden meget diversitet. I stedet skal performancekriterier indgå i den samlede designsyntese, som skaber en generativ designproces. Vigtigst af alt skal performancekriterierne inkorporeres i en proces, der skaber balance mellem kreativitet og effektivitet (Kolarevic 2005).

Performancebaseret design udnytter fakta og data som den primære designdriver, der gennem en parametrisk model skaber et system, der verificerer, validerer og evaluerer et design. Her er performancesimuleringer ikke kun en fase, hvor et design evalueres, men i stedet en motor der driver designets form.

> *"A determinant and method for the creation of architectural form. In such circumstances digital design diverges from a design paradigm in which the formal manipulative skills and preferences of the human designer externally control the process to one in which the design is informed by internal evaluative and simulation processes."* (Oxman 2008)

Aksamija og Zaki (Aksamija & Zaki 2010) beskriver forskellen mellem performancebaseret design og den traditionelle designmetode som følgende:

Traditionel Designmetode	Performancebaseret Design
Indeholder forenklede antagelser baseret på tommelfingerregler, som kan være unøjagtige (fx tvinge en æstetisk funktion til bygningen)	Bruger performance measures med faktiske kvantificerbare data og ikke tommelfingerregler
Ikke korrekt i forhold til performance meassures af designløsningen.	Forsøger at udvikle en "forenklet" model af et komplekst fysisk system
	Bruger modellen til at analysere og forudsige, hvordan bygningen vil opføre sig
	Producerer en mere realistisk vurdering af designet

Tabel 8-1 forskellen mellem den traditionelle designmetode og performancebaseret design

8.1 DESIGNOPTIMERING VS. DESIGNRATIONALISERING

Den traditionelle designproces anvender performancesimuleringer efter den formgivende designproces. Dermed er projektet allerede meget fastlåst, og det er svært at fortage andet end mindre ændringer i designet. Hvis performanceanalysernes resultater skal have en reel indvirkning på projektets udvikling, er det nødvendigt at inddrage dem allerede i formdesignet. Med det nuværende workflow er optimeringsprocessen meget besværlig, idet der skal opbygges en ny analysemodel efter hver ændring i designet. Denne proces skal forgå iterativt, hvor hvert resultat medfører nye ændringer, indtil det optimale design findes.

Ved at forbinde flere datakilder i den tidlige designproces kan flere beslutninger baseres på konkret data og fakta fremfor personlige holdninger og erfaringer. Det endelige arkitektoniske design kan altså gå fra at være en repræsentation af designerens holdninger, vilje og erfaringer til at være et produkt baseret på samspil mellem data og menneskelig viden (Anton & T□nase 2016). Greg Lynn (Lynn 1999) definerer design som et abstrakt rum, hvor reelle data fungerer som brændstof for designets formgivning. Gennem denne tilgang bestemmes designet af variabler og regler, der baseres på performance. Dertil skal designets form også omfattes af subjektive og objektive kriterier, der er relevante for det arkitektoniske koncept (Kolarevic 2005).

Designoptimering og designrationalisering er to processer, der begge bruger digitale værktøjer men alligevel er fundamentalt forskellige. Begge processers mål er at optimere et designforslag, mens forskellen findes i måden, de integrerer performancekriterier i projektet. Designoptimering er en metode, hvorved man forsøger at optimere en fastlagt form, der er udarbejdet på baggrund af æstetik ved at foretage performanceanalyser (Ceccato 2012). Softwaren, der bruges til at foretage analyserne, vil i dette tilfælde fungere som et værktøj og ikke en partner til designeren. Denne metode forsøger altså at forbedre effektiviteten af en fastlagt form.

Designrationalisering forsøger at vende processen om gennem en metode, der sammensætter geometriske regler med performance meassures for at genere en optimal løsning. Udgangspunktet er altså ikke længere en geometrisk form men i stedet en række parametre, der igennem regler og beregninger er forbundne (Anton & T□nase 2016). Derved fungerer analysesoftwaren ikke kun som et værktøj men i højere grad som en partner, der i samarbejde med designeren kan genere det mest optimale output. Analysesoftwaren kombineret med en parametrisk model skaber et system, der automatisk kan håndtere ændringer i designet. Alle resultater af analyserne føres direkte ind i et loop af modellens parametre, som derved opdateres automatisk og tæt på realtime. Gennem digitale værktøjer og generative designsystemer kan denne proces automatisk fortages iterativt, til man har dækket et ønsket antal alternativer fra modellens design space. Rationaliseringsmetoden skaber altså en proces, hvor der er et direkte link mellem performanceanalyser og den arkitektoniske form. Designprocessen ændres dermed fra formgivning til formanalysering, og design processen går fra et fast design til et mere procesorienteret design.

8.2 DESIGNING-IN PERFORMANCE

Bygninger, der performer bedre både hvad angår energi, CO2-udledning og generelt mere bæredygtige byggerier, er en vigtig del af nutidens dagsorden for byggeri. At opnå bedre performende byggerier starter allerede fra den første designtanke gøres og er i hele det videre projektforløb en vigtig faktor. Performancebaseret design er derfor en vigtig ting i byggeriet, da denne metode forsøger at tænke performance ind helt fra starten. Nye teknologier gør performancesimuleringer mere tilgængelige i de tidlige faser, og ideen om at integrere disse simuleringer med automatisering og designfeedback skaber perspektiver om bedre performende bygninger (Gerber & S. H. E. Lin 2014). Designing-in performance konceptet, som er opfundet af Shih Hsin Lin Eve og David Jason Gerber, er en del af performancebaseret design-feltet. Designing-in performance sætter fokus på at levere performancefeedback, der kan influere designbeslutninger på en måde, der igennem konventionelt design ikke er muligt. Metoden har især fokus på de tidlige designfaser, hvor designændringer stadig kan indarbejdes i projektet og har stor indflydelse på det endelige byggeri. Design-in performance er altså et designmiljø, hvor designeren kan få performancebaseret designfeedback direkte gennem den parametriske designmodel.

Udfordringerne ved at integrere design med performance measurments kan opdeles i to overordnede fokusgrupper. Det ene fokus er på interoperabilitetsproblemer blandt software og forskellige fagområder, mens bedre interoperabilitet mellem software vil lette generering og evalueringen af designalternativer, er det ikke tilstrækkeligt. Dette fører til det andet fokusområde, hvor der lægges vægt på at finde en intelligent søgemetode, der giver mulighed for hurtig evaluering til yderligere at understøtte designbeslutninger i de tidlige faser (Gerber & S. H. E. Lin 2014). I forbindelse med Lin og Gerbers andet fokusområde bliver det relevant at kigge på udviklingen af optimeringsteknikker som intelligente søgemetoder, der gør det muligt at frembringe de mest optimale designs. Anvendelsen af parametrisk modellering kombineret med optimeringsteknikker har trukket opmærksomhed som værende en potentiel løsning til at tilvejebringe en intelligent søgemetode til effektiv feedback. Multidisciplinary Design Optimization MDO betegner her metoden, der sammenkobler den parametriske model med optimeringsteknikker til at analysere flere fagdiscipliner i samme omgang.

Som respons på de observerede huller i den nuværende forskning inden for performancebaseret design og MDO, udviklede Lin og Gerbers (Gerber & S. H. E. Lin 2014) metoden Evolutionary energy performance feedback for design (EEPFD). EEPFD inkorporerer både konceptuelle energianalyser og designudforskning af simpel til

kompleks geometri for at give designeren performancefeedback i den tidlige beslutningsfase. EEPFD er udviklet ved hjælp af to væsentlige komponenter fra MDO-metodologi, parametrisk design og multiobjektiv optimering (MOO).

8.2.1 MULTIDICIPLINARY DESIGN OPTIMIZATION

Design er en tværfaglig proces, der hele tiden forsøger at balancere forskellige performancemål, der alle på hver sin måde er vigtige for det endelige produkt. I takt med at de teknologiske muligheder har udviklet sig og den øgede tilgængelighed af data, er designprocessen i byggebranchen blevet mere kompleks (Gerber & S.-H. E. Lin 2014). For at kunne håndtere den øgede kompleksitet og det øgede antal af performancemål, er der behov for en systematisk problemløsningsteknik. I mange design- og teknikområder er Multidisciplinary Design Optimization (MDO), blevet udforsket som en potentiel tilgang til at håndtere dette problem. MDO er en metode, der gennem analyser inden for en række fagområder forsøger at optimere et design. MDO giver designeren mulighed for at indarbejde alle relevante fagdiscipliner i en samlet analyse for på den måde at kunne simulere konsekvenserne af fagområdernes interaktion med hinanden. Dette er optimalt, idet designet kan optimeres i forhold til flere analyser på samme tid fremfor sekventielt at ændre designet på baggrund af enkelte faganalyser. Dog ses det at ved at inkludere flere fagområder i designanalyserne, at selve designproblemet bliver langt mere komplekst i forhold til enkelte faganalyser. MDO metoden bruges inden for en række industrier, herunder bildesign, elektronik, arkitektur, computere og elforsyning. Dog ses metoden mest brugt inden for luftfartsindustri i forbindelse med udvikling af fly og rumfartøjsdesign (Marler & Arora 2004).

8.3 GENETISKE ALGORITMER I DESIGN

En genetisk algoritme (GA) er en søgebaseret optimeringsteknik baseret på principperne om genetik og naturligt selektion. Det bruges ofte til at finde optimale eller næsten optimale løsninger på vanskelige problemer, som ellers ville tage meget langt tid at løse og til at løse optimeringsproblemer.

Wang et al. (Wang et al. 2005) beskriver i deres paper, hvordan genetiske algoritmer i højere grad bruges i forbindelse med bygningsdesign. Genetiske algoritmer er optimale i forbindelse med MOO analyser, fordi der kan lokaliseres flere Pareto[3]-optimale løsninger på en gang. Udforskningen af flere designalternativer er essentiel for at finde det optimale bygningsdesign, hvilket gør GA'er til en velegnet optimerings- og søgemetode i de tidlige faser.

Den genetiske algoritme blev udviklet til at løse problemer, hvor løsningsrummet er så stort, at en "brute force" -algoritme ville tage for lang tid. Et eksempel:

> *"Here's an example: I'm thinking of a number. A number between one and one billion. How long will it take for you to guess it? Solving a problem with "brute force" refers to the process of checking every possible solution. Is it one? Is it two? Is it three? Is it four? And so forth. Though luck does play a factor here, with brute force we would often find*

[3] En Pareto-optimal allokering eller Pareto-ligevægt er en tilstand, hvor ingen funktion kan opnå en bedre stilling uden, at en anden samtidig opnår en ringere stilling. Hvis der er Pareto-uligevægt, kan mindst én funktion opnå en bedre stilling uden, at det går ud over andre. I dette tilfælde kan man opnå en Pareto-forbedring ved at foretage den pågældende reallokering (fra: https://da.wikipedia.org/wiki/Pareto-optimal).

> *ourselves patiently waiting for years while you count to one billion. However, what if I could tell you if an answer you gave was good or bad? Warm or cold? Very warm? Hot? Super, super cold? If you could evaluate how "fit" a guess is, you could pick other numbers closer to that guess and arrive at the answer more quickly. Your answer could evolve."* (Shiffman 2012)

8.3.1 DARWINISME

Charles Darwin er en af historiens mest betydningsfulde forskere inden for forståelsen af livet på jorden. I 1859 udgav Darwin sin bog om arternes oprindelse, hvori han introducerede teorien omkring evolution gennem naturlig selektion. Teorien indeholdt to centrale elementer: *Evolutionspostulatet*, som siger, at alle arter ved gradvise ændringer er opstået fra en fælles stamform, samt *selektionspostulatet*, der beskriver, at gradvise ændringer inden for en art er frembragt ved naturlig selektion, hvor de bedst tilpassede individer i hver generation sætter det største antal afkom i verden.

For at evolution gennem naturlig selektion kan opstå, er det nødvendigt, at tre vigtige elementer er tilstede:

- **Arvelighed:** en proces der er nødvendig for, at en ny generation kan modtage den tidligere generations egenskaber.
- **Variation:** er nødvendigt for at opnå en udvikling inden for en art. Hvis ikke der er variation i befolkningen, vil alle fremtidige børn være nøjagtigt de samme som deres forældre.
- **Selektion:** er en mekanisme, der bestemmer hvilke befolkningsmedlemmer, der har mulighed for at nedarve deres gener til fremtidige generationer. Selektion sættes ofte i forbindelse med udtrykket "survival of the fittest". En sådan mekanisme sørger altså for, at de individer, der er bedst til at tilpasse sig det miljø, det befinder sig i, har den største chance for at reproducere og derved videregive sine gener.

Disse elementer fra Darwins teori om naturlig selektion er de samme principper, der udgør tankerne bag brugen af genetiske algoritmer. Ved at adoptere elementerne bag Darwins teori kan man hurtigere finde frem til løsninger, der er bedst i forhold til det miljø de testes i.

For at illustrere hvordan en traditionel GA virker, kan *"the infinity monkey theorem"* bruges *(https://en.wikipedia.org/wiki/Infinite_monkey_theorem)*. The infinity monkey theorem bygger på teorien om, at en abe, der tilfældigt taster på et keyboard i et uendeligt tidsrum, med sikkerhed vil have indtastet en given tekst såsom William Shakespeares komplette værker. Problemet med teorier som disse er, at sandsynligheden for, at aben rent faktisk får indtastet Shakespeares værker, er så lille, at selv hvis den startede ved universets oprindelse, ville det stadig være usandsynligt, at den havde ramt rigtigt.

Hvis aben taster på et keyboard med et reduceret antal taster med 27 karakterer, 26 bogstaver og en mellemrumstast, vil dette betyde, at sandsynligheden for, at en hvilket som helst karakter rammes, er 1/27. Hvis aben skal skrive sætningen "To be or not to be that is the question", er der 39 karakterer der skal rammes i rigtig rækkefølge. Chancen for at få det første karakter rigtigt er altså 1/27, og det samme gælder for de resterende karakterer. Chancen for, at alle karakterer tastes i den rigtige rækkefølge, kan derfor skrives som $(1/27)^n$, hvor n er antallet af karakterer i målsætningen.

Regnestykket bliver derfor $(1/27)^{39}$, hvilket giver en sandsynlighed på 1 ud af 66.555.937.033.867.822.607.895.549.241.096.482.953.017.615.834.735.226.163. Dermed er sandsynligheden for at ramme sætningen "To be or not to be that is the question" meget lav. Selv med en computer der kan indtaste 1 million sætninger i sekundet, ville det tage 9.719.096.182.010.563.073.125.591.133.903.305.625.605.017 år at opnå en sandsynlighed på 99% for at ramme den korrekte sætning. Dermed ville en "brute force" løsning være helt umulig i dette tilfælde, da selve universet har en estimeret levetid på 13.750.000.000 år (Shiffman 2012).

Med ovenstående eksempel i tankerne er det tydeligt, at en "brute force" metode ikke er optimal til at løse problemer, hvor den/de mest optimale løsninger skal findes. Genetiske algoritmer gør det muligt at opnå det samme resultat som "brute force" metoden men på en langt hurtigere og mere effektiv måde.

8.3.2 POPULATION

I GA feltet er en population altafgørende for at finde en løsning, der er tilfredsstillende inden for et givent område. En population er en gruppe af tilfældige optioner, der hver især består af en række gener, som definerer dem. For at kunne udvikle nye løsninger er det nødvendigt med variation i de individer, der er i populationsgruppen. Dette kan føres tilbage til Darwins teori, hvor variation er et af de tre elementer, der er nødvendige for at skabe udvikling. For at opnå variation i en population er det vigtigt, at den består af et vist antal individer, hvor antallet af individer kan variere alt efter problemets type og kompleksitet.

Problemet i *"the infinity monkey theorem"* gik ud på at finde en Shakespeare sætning, som bestod af flere forskellige gener (karakterer), der alle skal placeres på deres respektive korrekte pladser. Hvis man simplificerer dette problem til i stedet at skulle finde et simpelt ord såsom *"Kat",* er der brug for en indledende population, der kan udvikles på. Eksempelvis kunne startpopulationen bestå af følgende "ord":

- Hor
- Bnm
- Asf

Kigger man på denne population, er det tydeligt, at der ikke er nok variation, uanset hvordan man kombinerer bogstaverne, vil ordet *"Kat"* aldrig kunne generes. Var der derimod en population på 100 eller 1000 forskellige ord, ville sandsynligheden for, at hver karakter ville være på sin rigtige plads mindst én gang, være langt højere.

8.3.3 SELEKTION

Det tredje element i Darwins teori handler om udvælgelse eller *selektion,* hvor begrebet "survival of the fittest" gør sig gældende. Selektion handler i alt sin enkelthed om at udvælge, hvilke individer der skal bruges til at skabe den næste generation af mulige løsninger. Mere præcist handler det om at udvælge de mest egnede individer til at skabe nye individer, der er mere egnede end deres forældre ved at forene deres gener.

I genetiske algoritmer kan selektion opdeles i to faser. Første fase handler om at evaluere de enkelte individers fitness, hvilket sker på baggrund af en speciel *fitnessfunktion,* der tilegner hvert individ en numerisk værdi, som beskriver hvor egnet, individet er. Dette er et af de vigtigste elementer i en GA, hvor målet er at udvikle en optimal løsning på et problem.

Der findes mange måder at lave en fitnessfunktion på, og hvert problem kræver en unik fitnessfunktion. Det før beskrevne "gæt et ord"-problem kan igen bruges til at

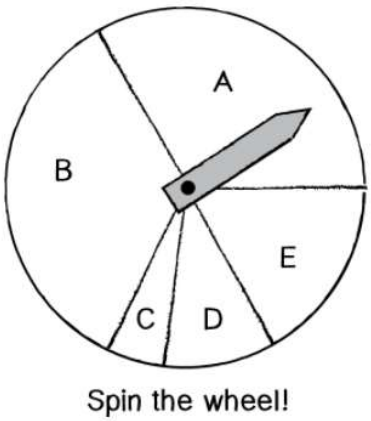
illustrere, hvordan en fitnessfunktion kan bruges. Målet med problemet er at genere et givent ord, der består af en række karakterer, og det ultimative mål er altså at placere de rigtige karakterer på deres rigtige pladser. Fitnessfunktionen for dette problem er derfor simpel: *fitness = antal korrekte karakterer på den rigtige plads*

Individ	Fitness
Kol	1
Vbn	0
Rat	2

Tabel 8-2 Fitnessfunktion for "Gæt et ord"-problemet

Anden fase består i at genere en *'Mating Pool'*. En mating pool er en gruppe, hvor de individer, der er egnet til at blive forældre, placeres. Typisk arbejdes der med to metoder til at placere forældre i en mating pool, enten *elite* metoden eller *probability* metoden. Den første er den mest simple og benytter kun de bedste individer til at lave et nyt *'child'*, men problemet med denne metode er, at den ofte er på bekostning af variationselementet. *Probability* metoden tager en lidt anden tilgang i forældreudvælgelsen. I stedet for at bruge de direkte fitnessværdier normaliseres værdierne først ved at tage den samlede sum af værdier og derefter dividere hver score med den samlede sum. Daniel Shiffman (Shiffman 2012) illustrerer denne metode med et lykkehjul, hvor det enkelte individs feltstørrelse afhænger af hvor høj en fitness, den har opnået.

Med "lykkehjul"-metoden vælges to tilfældige forældre dog med større sandsynlighed for et individ med en høj fitness score. På denne måde skabes der langt højere variation end ved eliteudvælgelsesmetoden.



Parent	Probability
A	30%
B	40%
C	5%
D	10%
E	15%

Figur 8-1 Selektion probabilistisk metoden, illustreret som et lykkehjul (Shiffman 2012)

8.3.4 GENERATIONSEVOLUTION

Gennem selektionsdelen af en genetisk algoritme udvælges de individer, der sammen skal producere den næste generation, og her kommer Darwins arvelighedselement ind i billedet. Nye individer arver egenskaber fra deres forældre fra den tidligere generation, hvilket giver dem en ny DNA-sammensætning, og derved skabes der en generationsevolution. I en genetisk algoritme skabes denne evolution typisk gennem to skridt. Det første skridt kaldes ***crossover***, som er en proces, hvor gener fra de to forældre blandes for at skabe et nyt individ.

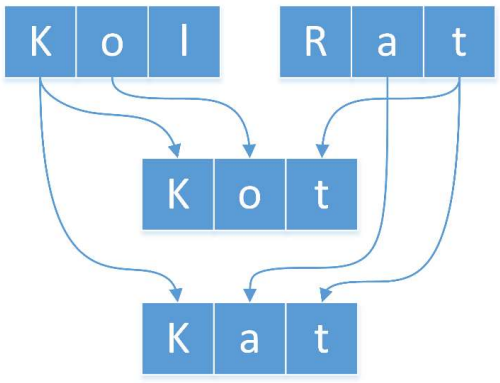
I crossover bruges en del af et individ og en anden del fra et andet individ. I det tidligere brugte ”gæt et ord”-eksempel ville to individer udvælges, hvorefter der foretages crossover. De to individer kunne for eksempel være

Individ A = Kol

Individ B = Rat

Ud fra disse individer kan der nu skabes et nyt barn, der deler gener fra hvert forældreindivid. Der findes flere måder at skabe crossover på i en genetisk algoritme.

50/50 metoden tager 50% af individ A’s gener og 50% af individ B’s gener. Ved at bruge 50/50 metoden på individ A og B ville det nye barn enten komme ud som ”Kot” eller ”Kat”.



Figur 8-2 Crossover 50/50 eksempel

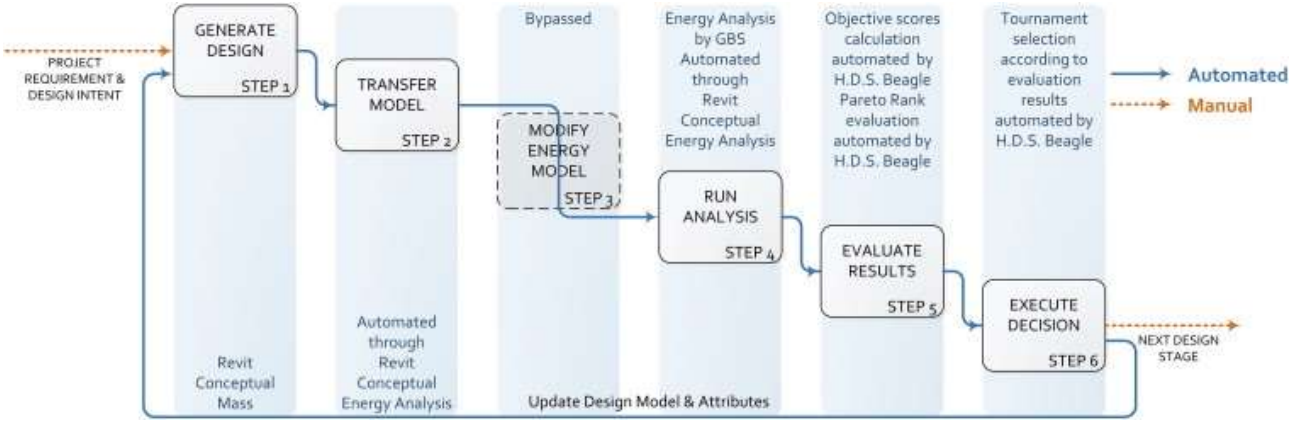
En anden måde at lave crossover på er ved at buge en tilgang, hvor det tilfældigt bestemmes hvor mange gener, der tages fra individ A, mens det resterende antal gener tages fra individ B.

Andet skridt i udviklingen af en ny generation kaldes ***mutation***, og dette element tilføjes som det sidste, inden et nyt barn tilføjes til den nye generation. Mutation er ikke nødvendigt for en GA, og i tilfælde med store populationer kan mutation være overflødigt. Styrken ved mutation er, at den tilføjer endnu mere variation imellem generationerne, som er essentielt i en populations udvikling. Mutation i en GA beskrives som et %-tal eller en sandsynlighed for, at mutation sker. Hvis mutationssandsynligheden er 5%, ville der være 5% chance for, at et bogstav i det nye barn laves om til et andet tilfældigt bogstav. Mutationssandsynligheden er en fin balance, hvor for meget mutation vil resultere i at de stærkeste gener ikke føres videre, og for lidt kan betyde variationen ikke er stor nok til at kunne udvikle bedrer generationer.

8.4 EEPFD SOM PERFORMANCEBASERET DESIGNMETODE

EEPFD er udviklet som en metode til en specifik prototype, der inkorporer forskellige software for at lave et optimalt energidesign. Selve processen og metoden, der er brugt til at udvikle prototypen, er interessant at undersøge yderligere i forhold til at bruge i projektets videre arbejde. En overordnet gennemgang af Lin og Gerber's projekt vil blive givet i det følgende for at få indsigt i deres metode til et performancebaseret design workflow.

Processen i EEPFD som designing-in performancemetode beskrives af Lin og Gerber i seks trin, som illustreret i Figur 8-3. I det første trin generes den indledende geometri gennem et parametrisk design, der gør det muligt at udforske designalternativer. Den parametriske model indeholder regler, der styrer geometrien i forhold til de inputs, der gives. Genereringen af designalternativer er en del af den automatiserede proces, der drives af en tilpasset genetisk algoritme (GA). Når først det indledende design er indført i det automatiske system, gennemføres de efterfølgende trin i Figur 8-3 indtil automatiseringsloopet afbrydes af brugeren, eller hvis systemet møder et afbrydelseskriterie. Når automatiseringsloopet er færdig, er der to måder at gå videre på: 1) et designalternativ er valgt ud fra MOO analysen, og designet fortsætter til næste udviklingsfase eller; 2) brugeren gennemfører manuelt ændringer i den oprindelige model eller modellens regelsæt, før automatiseringsloopet forsættes.



Figur 8-3 EEPFD's sekstrinsproces for et integreret designing-in performance workflow

8.4.1 MÅLFUNKTIONER

I EEPFD metoden opsættes der forskellige målfunktioner, som bliver styrende for det endelige design. I Lin og Gerbers EEPFD projekt er der opsat tre målfunktioner, der bruges til at måle og evaluere præstationen af individuelle potentielle designløsninger. I det specifikke projekt er de tre målfunktioner henholdsvis rumprogram overholdelse (RPO), der evaluerer hvor godt designet møder de opsatte rumkrav, energiforbrug (EF), der estimerer designets energiperformance og til sidst den finansielle performance (FP), der evaluerer designet i forhold til finansielle perspektiver.

De udvalgte målfunktioner vurderes enkeltvis i forhold til det samlede designforslag ved at opstille funktionerne, så de kan indgå i en samlet formel. I Lin og Gerbers projekt er de tre funktioner opstillet som følgende:

- R_{obj} = Max. RPO
- E_{obj} = Min. EF
- F_{obj} = Max. FP

Yderligere forklaring til de tre målfunktioner vil ikke blive givet i dette projekt, da dette blot er eksempler på opsætningen af målfunktioner i EEPD's metoden.

8.4.2 DESIGN GEOMETRIPARAMETRE

Geometriparametre bruges til at beskrive metodens parametriske model i forhold til alle de formgivende parametre og deres respektive intervaller. Den parametriske models fleksibilitet og evne til at genere forskellige designs afhænger af brugeren og antallet af parametre i modellen. Geometriparametrene kan opdeles i tre kategorier:

- Drivende Parametre
- Drevet Parametre
- Faste Parametre

Drivende parametre betragtes som uafhængige parametre med acceptable værdiintervaller, f.eks. bygningshøjde, antal etager, vridningsgrad i designet eller orientering. Drevne parametre er de parametre, der i sig selv ikke har fastlagte intervaller, men i stedet er styret af de drivende parametre. Faste designparametre besidder kun en enkelt given værdi og er ikke afhængig af hverken drevne - eller drivende parametre.

8.4.3 GENETISK ALGORITMEKODNING

Genetiske algoritmer arbejder ofte med et binært kodningssystem, primært fordi de første forsøg med GA'er brugte et binært kodningssystem. Værdikodning er et andet kodningssystem, der kan bruges i problemer, hvor der arbejdes med komplicerede værdier såsom reelle tal. Brug af binær kodning til denne type problemer ville være meget vanskelig (Obitko 1998).

I forbindelse med designudforskning er det mest hensigtsmæssigt at benytte værdikodningssystemet, da dette i højere grad leverer den fleksibilitet, der kræves af et performance based workflow. Derudover giver værdikodningssystemet designere mulighed for at formulere deres designproblemer ved brug af en standard designplatform og parametrisk modelleringsproces.

Gennem værdikodningsskemaet kan alle drivende parametre, der består af et brugerdefineret interval af acceptable værdier betragtes som et *gen*. Et *kromosom* defineres derefter som en serie af gener med en værdi, der falder inden for det acceptable interval tildelt til hvert gen. Derfor vil et design med 13 drivende parametre som beskrevet i Tabel 8-3, have designalternativer med et kromosom bestående af 13 værdier, en værdi tildelt for hvert gen.

Værdikodningsskemaet benytter tre forskellige datatyper: reelle tal[4], heltal[5] og enumerations[6]. Reelle tal kan betragtes som tal, hvis værdier er nøjagtigt som genereret af GA'en uden nogen ændring. TwistAngle-parameteret, som er angivet i Tabel 8-3, er defineret som et reelt tal med et interval fra 0 ° til 90 °, og derfor vil enhver værdi, der genereres gennem enten mutation eller crossover, som falder inden for dette interval, betragtes som gyldig, uanset om det er 54,3 ° eller 45 °.

[4] Reelle tal er en samling af alle forskellige talklasser, klassen indeholder derfor både naturlige tal, heltal, rationelle og irrationelle tal
[5] Heltal dækker over alle positive og negative naturlige tal
[6] I computer programmering er en enumerated type en datatype bestående af et sæt navngivne værdier kaldet elementer eller medlemmer. En enumeration er en komplet liste over alle elementer i en samling.

Heltalværdier er imidlertid begrænset til hele tal (integers) på trods af den faktiske værdi genereret af GA'en. For eksempel har parameteret "Level#Hotel" i Tabel 8-3 en intervalværdi mellem 4 og 6. Hvis en værdi på 5,3 opnås under GA-operationen, vil en værdi på 5 blive brugt som værdien for dette parameter, da kun heltalværdier er tilladt i denne kategori. En lignende fremgangsmåde anvendes til en enumeration af tekstværdier (strings). En enumeration indeholder værdier, der i stedet for at anmode om en værdi anmoder om, at et valg skal foretages fra en tekstbaseret liste. Eksempelvis kan et enumerationparameter kræve, at en konstruktionstype vælges fra den tekstbaserede liste. I disse tilfælde tildeles en heltalsværdi til hver parameterværdi, og heltallet behandles således af GA-operationen. Når operationen er fuldført koordineres den resulterende heltalværdi med det originale tekstbaserede valg. Hvis der for eksempel er 10 valg af konstruktionstyper til rådighed, og en værdi på 3,2 opnås gennem GA-operationen, vil det 3. valg på konstruktionstypelisten blive tildelt denne værdi.

	Parameter Name	Type	Unit	Exploration Range
G1	Level#Hotel	Integer	N/A	[4,6]
G2	Level#Office	Integer	N/A	[4,6]
G3	Level#Retail	Integer	N/A	[4,6]
G4	Level#Parking	Integer	N/A	[1,3]
G5	TwistAngle	Real	°	[0, 90]
G6	TopSetback	Real	ft	[1,10]
G7	ScaleFactor	Real	N/A	[0.8, 1.25]
G8	CanyonWidth	Real	ft	[12,30]
G9	BaseSetback	Real	ft	[6,15]
G10	Target Percentage Glazing	Real	N/A	[0.2, 0.83]
G11	Shade Depth	Real	ft	[0, 4.5]
G12	Target Percentage Skylight	Real	N/A	[0, 0.45]
G13	Conceptual Construction – Mass Glazing	strings	N/A	1. Single Pane Clear — No Coating 2. Single Pane — Tinted 3. Single Pane — Reflective 4. Double Pane Clear — No Coating 5. Double Pane — Tinted 6. Double Pane — Reflective 7. Double Pane Clear — LowE Cold Climate, High SHGC 8. Double Pane Clear — LowE Hot Climate, Low SHGC 9. Double Pane Clear — High Performance, LowE, High Tvis, Low SHGC 10. Triple Pane Clear — LowE Hot or Cold Climate

Tabel 8-3 Eksempel på parametre, deres datatype og deres interval af acceptable værdier. Eksemplet er fra Gerber og Lin's EEPFD projekt og viser kun tabel for "Design Geometry Parameters", lignende tabel er lavet for projektets andre målfunktioner.

8.4.4 DESIGNEVALUERING

Alle genererede designalternativer evalueres i forhold til de tidligere beskrevne målfunktioner for at finde det optimale design. De enkelte målfunktioner er ofte i direkte eller indirekte konkurrence med hinanden forstået på den måde, at forbedringer i en målfunktion kan have negativ indvirkning på en anden målfunktion. Jagten på det optimale design bliver derfor et spørgsmål om at finde det bedste kompromis, hvor alle målfunktioner optimeres så meget som muligt fremfor at finde en enkelt optimeret løsning (Veldhuizen et al. 2007). Derfor kan en Pareto-rangeringsmetode anvendes som evalueringsmekanisme til at finde de optimale designkompromisser i et givent design space. Pareto-metoden benytter *Pareto-Dominance (p<)* konceptet til at sammenligne to alternativer og udvælge det bedste design. Et alternativs overlegenhed over et andet alternativ bestemmes ved at sammenligne de to alternativers performance på tværs af de opsatte målfunktioner.

Definitionen af *Pareto-Dominance* kan derfor opskrives som nedenstående formel, der tager udgangspunkt i Lin og Gerbers EEPFD projekt og dets tidligere beskrevet målfunktioner:

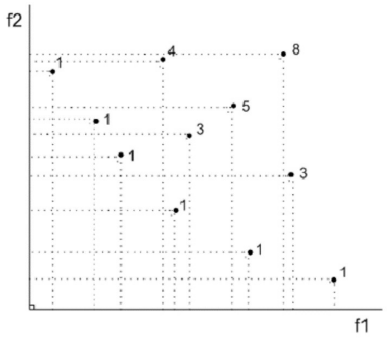
$$\forall f \in \left\{ s_{obj}, E_{obj}, F_{obj} \right\} f(\text{solution}_1) \leq f(\text{solution}_2)$$
$$\exists f \in \left\{ s_{obj}, E_{obj}, F_{obj} \right\} f(\text{solution}_1) < f(\text{solution}_2)$$
$$\Rightarrow \text{solution}_1 \quad p< \quad \text{solution}_2.$$

Definitionen beskriver, at hvis solution_1 har bedre performance i de tre målfunktioner end solution_2 (betegnet solution_1 p< solution_2), er solution_1 det dominerende alternativ i forhold til rangering. Eksempelvis hvis alternativ A har en målfunktionsscore på (94, 160, 65), og alternativ B har en score på (97, 102, 82), så vil B anses som værende det dominerende alternativ og derved rangere højere end A. Hvis alternativ C har en score på (90,104, 85), mens alternativ D scorer (98, 153, 90), vil det betyde at C og D betragtes som ude af stand til at dominere hinanden. I denne situation sker der det, at D har en bedre score i henholdsvis R_{obj} (s_{obj}) og $F_{obj,}$ mens C score bedre i E_{obj}. Eftersom alle målfunktioner prioriteres ens, er det hverken muligt for D eller C at dominere den anden, og derfor tildeles de den samme rangering.

Rangering af alternativer sker på baggrund af følgende formel:

$$\text{Rank}_i = 1 + \text{Num}(\text{Individual}_{\text{dominated}})$$

Alle alternativer tildeles som udgangspunkt rangeringen 1, hvis der efterfølgende kommer nye alternativer, der dominerer dem, tillægges antallet af dominerende alternativer til den nuværende rangering (se Figur 8-4). Derved vil alternativer med den bedste score rangeres som 1.

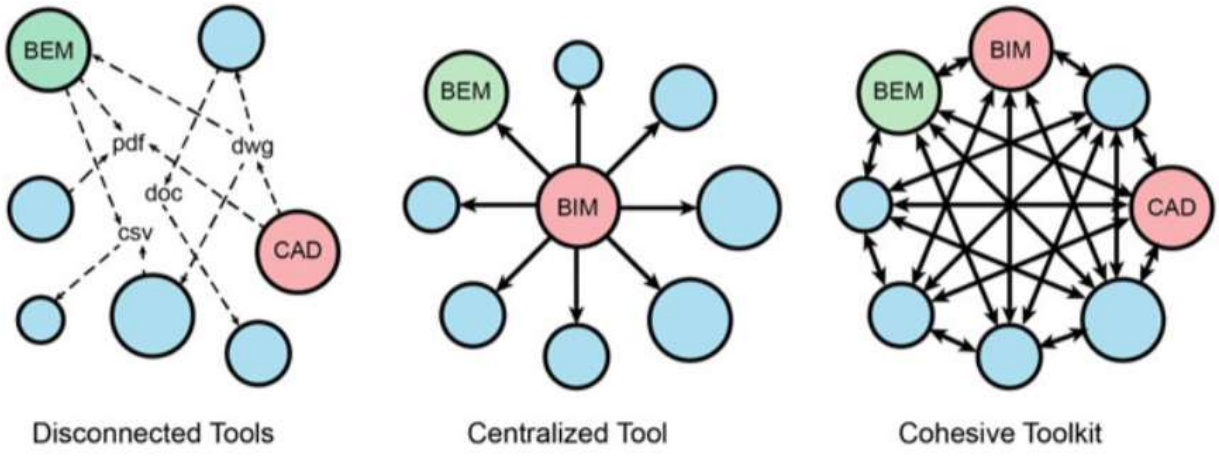


Figur 8-4 Multiobjektiv rangering

9 SOFTWAREVÆRKTØJER VS. SOFTWAREVÆRKTØJSKASSER

De foregående tre afsnit har givet en introduktion til hvilke elementer, der udnyttes i projektets tekniske løsning. Udover disse tre elementer er filosofien om at bruge software som en værktøjskasse, der indeholder forskellige komponenter, der kan sammensættes for bedst muligt at løse en opgave fremfor at bruge enkelte isolerede værktøjer ligeledes en vigtig del af dette projekts endelige løsning. Følgende afsnit vil give en introduktion til fordelene ved at kunne sammensætte komponenter fra en softwareværktøjskasse, og ligeledes vil nogle af den endelige løsnings komponenter blive introduceret.

Den digitale transformation, som byggebranchen gennem de sidste årtier har gennemgået, har gjort det mere almindeligt og anerkendt, at nye teknologier er nødvendige for at skabe bedre bygninger og designprocesser. I dag er der stadig uenighed i branchen om, hvordan nye teknologier bedst integreres i branchens workflows for at skabe mest mulig værdi. To typer software har hidtil været mest benyttet; isolerede software der er svære at integrere i et samlet workflow eller centraliserede software, der primært arbejder med envejskommunikation til andre værktøjer (Mackey & Roudsari 2018). Specielt når man kigger på performanceanalysesoftware, kan den isolerede softwaretype ses, hvor der findes en lang række specialiserede værktøjer indenfor performanceanalyser. I dag findes der værktøjer, der specialiserer sig i alt fra ventilationsdimensionering til akustiske analyser. I mange tilfælde ses det, at modeller skal ændres eller tilpasses i analyseværktøjet, fordi kommunikation mellem modellens platform og værktøjet ikke er god nok, hvilket bliver problematisk, når flere og flere værktøjer tilkobles processen. Problemet med isolerede værktøjer har været forsøgt løst mange gange, hvor nok det bedste forsøg er brugen af Building Information Modeling (BIM), der forsøger at centralisere data i en samlet model. Disse modeller bliver hurtigt meget tunge og svære at arbejde med, netop fordi de skal indeholde så store mængder data, hvilket gør dem ufleksible og dårlige at teste alternative løsninger på. Forskellige analyser kræver forskellige data, og derfor bliver det problematisk, hvis en samlet model skal indeholde data om alle analyser, og hvert element i modellen vil skulle indeholde en lang række egenskaber.



Figur 9-1 Isolerede, centraliseret og værktøjskasse software (Mackey & Roudsari 2018)

Branchen begynder i højere grad at indse, at hverken den isolerede tilgang eller ”et værktøj til det hele”-tilgangen er den rigtige til at løse byggebranchen softwareintegrationsproblemer (Mackey & Roudsari 2018). En balance mellem de to tilgange ses af flere som den bedste løsning på disse problemer. Mackey & Roudsari (Mackey & Roudsari 2018) påpepeger, hvordan både den isolerede og centraliserede softwaretilgang lider under den samme filosofi – begge har fokus på selve værktøjet fremfor workflowet, de skal indgå i og intergrationen mellem værktøjer.

En håndværker vil i langt de fleste tilfælde ikke kunne gøre sit job tilfredsstillende, hvis kun han havde ét værktøj til hvert job. I stedet har han brug for hele sin værktøjskasse, der er fyldt med specialiserede værktøjer, der hver især klarer en lille del af den samlede opgave, han skal løse. Det samme er gældende i softwareverdenen; et værktøj er sjældent nok, ligesom mange værktøjer, der ikke indgår i det samme workflow, ofte skaber et dårligt resultat.

Den øgede opmærksomhed på at skabe værktøjskasser, der understøtter hele workflows fremfor blot at fokusere på isolerede værktøjer eller centraliserede ”en platform til alt”-tilgange, er i god overensstemmelse med den øgede popularitet inden for visuel programmeringssprog (VPL). VPL tager forskellige funktioner og bryder dem ned i små komponenter eller noder, som man kan sammensætte på kryds og tværs for at skabe et workflow, der passer den situation, man er i.

9.1 SPECIALISERET FUNKTION

Kendetegnet ved en succesfuld softwareværktøjskasse er, at de værktøjer værktøjskassen indeholder specialiserer deres funktion på en specifik opgave. Det ses ofte, at de mest værdifulde værktøjer, man bruger i hverdagen, også er de mest simple, som løser en specifik opgave. Denne tankegang gælder i høj grad også, når der snakkes om de forskellige VPL, grundet de mange plugins eller tillægspakker, er det nødvendigt at skille sig ud ved at fokusere pakkens funktion og samtidig gøre det bedre, end hvad der ellers er tilgængeligt.

Selvom software, som er en del af en værktøjskasse, skal fokusere på én ting, er interoperabilitet med andet software i værktøjskassen ligeså vigtigt. Eksport- og importfunktioner er vigtige og især kompatibilitet til standardformater, så der sikres interoperabillitet med størst muligt omfang af andet software. Ofte fokuserer softwareleverandører på importfunktionalitet frem for eksportmuligheder, hvilket ofte gøres for at holde kunder i deres softwaremiljø. Ofte er dette dog ikke en god ide, da softwareleverandøren i stedet skal bruge tid på at udvikle funktionalitet, der allerede er tilgængeligt andre steder (Mackey & Roudsari 2018).

Et godt eksempel på et værktøj, som understøtter principperne om værktøjskasser frem for værktøjer, er Ladybug Tools (Ladybug Tools 2017). Ladybug Tools specialiserer sig inden for analyser relateret til klima og vejrdata og dækker inden for dette område over et bredt spektrum. Ladybug Tools fokuserer på det, de er gode til - at analysere data, og softwaren tilbyder derfor ikke andre ting uden for analysespektret, men gør det i stedet muligt at samarbejde med andet software.

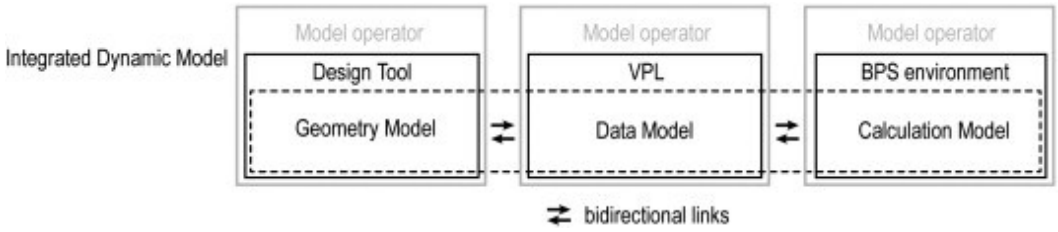
9.2 LADYBUG + HONEYBEE

Projektering af et byggeprojekt forløber over mange forskellige faser, iterationer og fokusområder. På trods af dette ses det, at mange af de tilgængelige analysesoftware kun kan udføre én analyse ad gangen, dette gør analyseprocessen ufleksibel. Udviklingen af et bygningsdesign kræver mange iterationer, og denne iteration og udvikling betyder, at der hurtigt opstår mange forskellige bygningsformer, hver form skal analysere, hvilket kræver forskellige modellerings- og analysesoftware.

Analyseprogrammer har tendens til at fokusere på bygninger, hvis design allerede er fastlagt og er derfor ikke stærke, når det gælder design space-udforskning (Roudsari 2016).

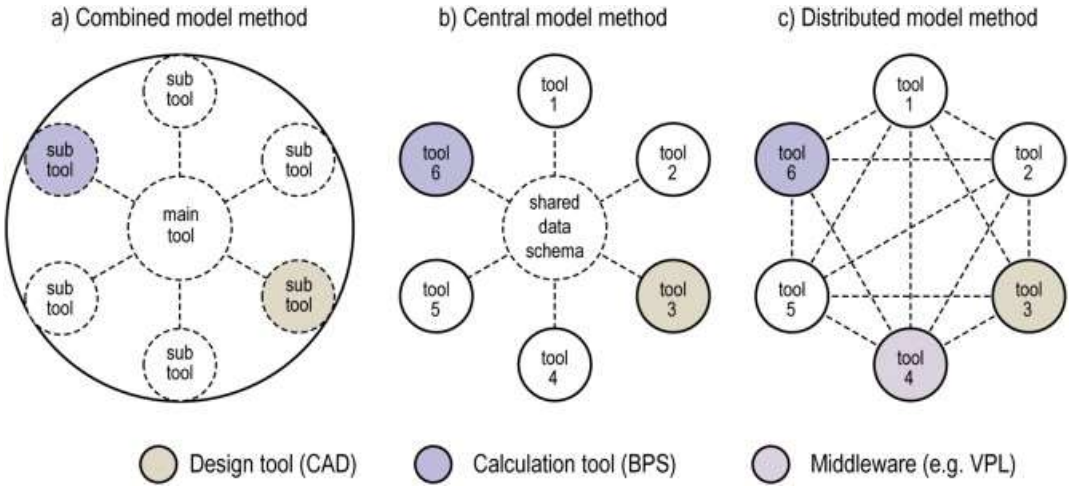
De seneste år er der sket en udvikling i, hvordan projektering og modellering af bygningsdesign sker, henholdsvis *'Computational Modeling'* eller parametrisk design, gennem visuel programmering og *'Building Information Modeling' (BIM)*. Tilsammen har disse modelleringsmetoder skabt nye muligheder for at analysere på bygningsdesignsperformance. Computational Modeling har især styrket mulighederne for Generative Design og gjort flere designiterationer mere tilgængelige på kortere tid. BIM baserede workflows har tilføjet mere data til modellernes geometri, hvilket gør flere og mere præcise analyser muligt. Disse nye modelleringsmetoder har skabt bedre mulighed for at understøtte design space-udforskning gennem iterative analyser ved at skabe tovejsforbindelser mellem modelleringsværktøjer og analyseværktøjer.

Negendahl (Negendahl 2015) beskriver den nye integration mellem modellerings- og analyseværktøjer som en *'Integrated Dynamic Model'*, hvor forbindelse mellem værktøjerne skabes gennem et visuelt programmeringsmiljø (se Figur 9-2). Det visuelle programmeringsmiljø gør det muligt for designere at tilpasse og automatisere analyse- og modelleringsprocessen i forhold til den fase, projektet er i.



Figur 9-2 Integrated Dynamic Model diagram (Negendahl 2015)

Integrated Dynamic Models er en modulbaseret model beskrevet af Negendahl (Negendahl 2015) som *Distributed model method*, hvor brugerne kan kombinere forskellige moduler, hvilket giver en høj fleksibilitet i analysearbejdet, og gør det muligt at integrere data fra forskellige værktøjer. Modultankegangen benytter *'middelware'*-komponenter til at transformere og manipulere data mellem de forskellige værktøjer, visuel programmeringsmiljøer fungerer i denne model som middleware (se Figur 9-3).



Figur 9-3 Koblingsmetoder mellem design og analyseværktøjer (Negendahl 2015)

Ladybug Tools har igennem de senest år opbygget flere integrated dynamic models, blandt de mere populære er Ladybug og Honeybee, som er open source plugins til VPS platformene Grasshopper og Dynamo. Begge plugins bygger på en Distributed model, som assisterer designere med at evaluere og udforske miljøperformance i et parametrisk modelleringsmiljø. Ladybug importerer EnergyPlus vejrfiler (.epw) i Grasshopper og Dynamo, ligeledes bruges Ladybug til at visualisere forskelligt 3D grafik i forbindelse med analyser. Honeybee integrerer simuleringssoftware i Grasshopper og Dynamo. Honeybee benytter kendte åbne analysemotorer inden for performancesimuleringer såsom EnergyPlus, Radiance, Daysim og OpenStudio til blandt andet at evaluere bygningers energiforbrug, dagslys og lysforbrug.

Ved hjælp af Ladybug Tools plugins skabes der en dynamisk kobling mellem VPS miljøet, validerede miljø datasets (EnergyPlus vejr filer) og forskellige simuleringsmotorer. Eftersom både Laybug og Honeybee er udviklet til 3D modelleringsværktøjer, gør de det muligt for designeren at bruge den originale designmodel som grundlag for analyserne. Forskellen mellem softwareværktøjskasser som Ladybug Tools og traditionelle ”black box” software er, at designeren selv kan styre og tilpasse simuleringsworkflowet. Selvom der i de fleste tilfælde skal bruges mere tid på at opsætte workflowet, fordi det skal programmers, vil den tid blive tilbagebetalt i potentialet for at køre scriptet mange gange og derved skabe langt flere iterationer af designet (Roudsari 2016).

10 STATE OF THE ART

I forbindelse med projektets endelige løsningsforslag skal alle de tidligere beskrevne elementer sammensættes, så de danner et workflow, der skaber et bedre analysegrundlag i de tidlige faser. For at kunne skabe dette workflow er det nødvendigt at have platform, hvor elementerne kan forbindes. Følgende afsnit vil give en introduktion til visuel programmering samt introducere to af de mere populære visuel programmeringssprog inden for parametrisk design.

10.1 VISUEL PROGRAMMERING

I løbet af de seneste år er Visual Programmings Sprog (VPS) indenfor bygningsdesign blevet mere og mere etableret som et værdifuldt redskab. Kendte softwareprodukter indenfor byggebranchen har med stor succes integreret VPS i deres programmer, især plug-ins som Grashopper til Rhinoceros3D, Dynamo til Autodesk Revit (og som selvstændig applikation) og Generative Components til Bently.

Et visuelt programmeringssprog (VPS) er et programmeringssprog, der giver brugeren mulighed for at oprette scripts ved at sammensætte grafiske komponenter i stedet for den normale tekstbaserede programmering. Et VPS gør programmering mere tilgængeligt for folk uden programmeringserfaring gennem visuelle udtryk og grafiske symboler. For eksempel er de fleste VPS'er (kendt som dataflow eller diagrammatisk programmering) baseret på en notation, der består af bokse og linjer, hvor hver boks behandles som enhed, der forbindes med pile, linjer eller buer, der repræsenterer relationer.

VPS findes i mange afskygninger og bruges inden for mange forskellige områder alt fra business intelligence til parametrisk design. VPS kan altså bruges i flere forskellige fokusområder. I høj grad bruges det til den mere hardcore databehandlingsdel, men der har de seneste år været stor udvikling inden for VPL og parametrisk design. Fælles for alle VPL'er er deres grafiske userinterface, der som nævnt i de fleste tilfælde består af blokke og relationer.

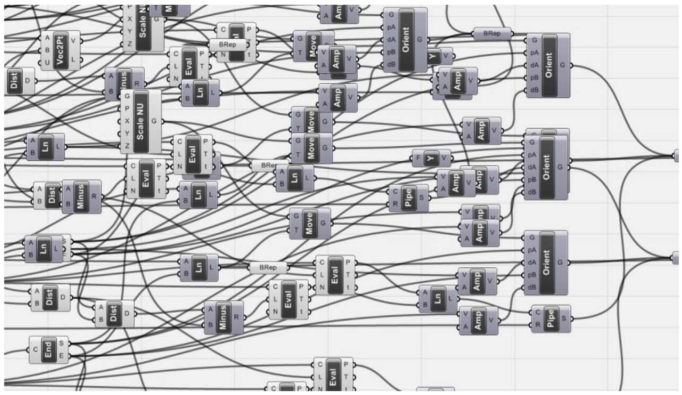
I forbindelse med udbredelsen af VPS til designsoftware er en række applikationer, der benytter VPS, de seneste år kommet på markedet. Nedenstående giver en beskrivelse af to af de mest populære VPS inden for byggebranchen.

10.1.1 GRASSHOPPER

Grasshopper er en visuel programmeringsplatform, som er udviklet som et plugin til det populære designmodelleringsværktøj Rhinoceros 3D. Grasshopper blev skabt som en udvikling af Rhinoceros 3D's "Record History"-funktion, som gjorde det muligt at tildele specifikke funktioner til geometri (Akos & Parsons 2014). I 2008 begyndte udviklingen af "Record History"-funktionen for at opnå mere frihed og kontrol over geometrien. Denne udvikling førte til pluginnet Grasshopper som et VPS til Rhinoceros 3D. Grasshopper kræver ingen tidligere programmeringserfaring, det benytter komponentblokke og relationslinjer, der tilsammen linker geometri med parametre og forskellige funktioner. Grasshopper er som de fleste VPS et dataflow programmeringssprog, hvor scripts i Grasshopper eksekveres i rækkefølge fra venstre til højre, og resultatet af scriptet produceres i Rhinoceros 3D.

Grasshopper er med tiden blevet meget populært blandt arkitekter grundet dets brugervenlighed og det, at det er så enkelt at lære. Der er dog stadig ulemper ved brugen af Grasshopper, hvor store scripts hurtigt bliver meget uoverskuelige (Figur 10-1) grundet de mange forbindelser der opstår, hvilket kan gøre det svært at forstå, hvordan de enkelte komponenter er forbundne. Dette gør det også ekstremt svært at

lave ændringer i modellen, hvilket er i overensstemmelse med Davis (Davis 2013) fem problematikker forbundet med brugen af parametriske modeller.

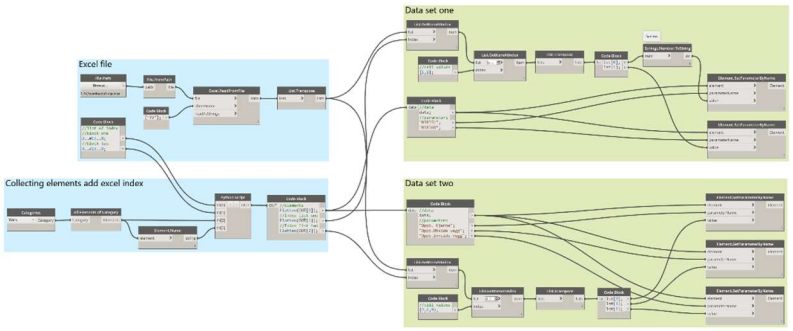


Figur 10-1 Grasshopper komponenter og forbindelser, uoverskuelighed i store scripts (Davis 2013)

Grasshopper har I høj grad været en drivende faktor inden for parametrisk og generativt design. I dag er det stadig et af de mest benyttede værktøjer, når det gælder computational design. På trods af den store popularitet ved Grasshopper er den helt store ulempe stadig, at det primært er værdifuldt i begyndelsen af et projekt, ,men når projektet bevæger sig længere frem, er det ikke muligt at detaljere Rhino/Grasshopper modellerne nok.

10.1.2 DYNAMO

Dynamo er et VPS udviklet af Autodesk som et plugin til Revit og som en standalone computational design platform. Ligesom Grasshopper gør dynamo det muligt for ikke-programmører at udvikle scripts ved at benytte komponent nodes og forbindelseslinjer. På mange måder ligner Grasshopper og Dynamo hinanden, deres userinterface principper er langt hen ad vejen ens, og begge benytter dataflowprincippet. Den helt store forskel mellem de to er den bagvedliggende motor. Mens Grasshopper benytter Rhinoceros 3D til geometrivisning, bygger Dynamo på Revits geometrimotor. Der er fordele og ulemper ved, at Dynamo bygger på Revit, den helt store fordel er, at Revits API kan tilgås direkte gennem Dynamo, hvilket gør det muligt at udvikle workflows, der benytter API'en til at automatisere opgaver i Revit. Ulempen er, at Revit i flere år har haft problemer, når det kommer til kompleks geometri og generelt performance. Dynamo har i høj grad gjort det mere tilgængeligt at generere kompleks geometri i Revit, dog går det ofte udover performance, hvilket betyder, at et script kan tage lang tid at køre. Dynamo kan helt overordnet opdeles i to hovedfunktioner, Revit automatisering og computational design.



Figur 10-2 Eksempel på opbygning af en Dynamo Graf

III. LØSNING

11 BESKRIVELSE AF LØSNING

Projektets løsning bygger på tanken om at skabe et workflow, der giver et mere datadrevet beslutningsgrundlag i den tidlige designfase. Løsningen bygger på de elementer der er beskrevet i rapportens hovedafsnit, og selve opbygningen af løsningen er foretaget ud fra den tidligere beskrevne *Designing-in performance*metode. Følgende afsnit vil beskrive den tekniske side af det udviklede workflow. Til løsningen er Dynamo brugt som udviklingsmiljø, hvilket primært er gjort grundet allerede eksisterende kompetencer inden for dette miljø. Andre VPL miljøer kunne ligeledes være brugt, og Grasshopper ville ligeledes være et oplagt valg, da det lever op til de behov, der er til workflowet. Muligheden for at benytte softwareværktøjskasser som Ladybug Tools samt have friheden til at kunne udarbejde tekstbaserede scripts har været blandt de højst prioriterede behov for workflowet. Begge disse behov er understøttet af både Dynamo og Grasshopper.

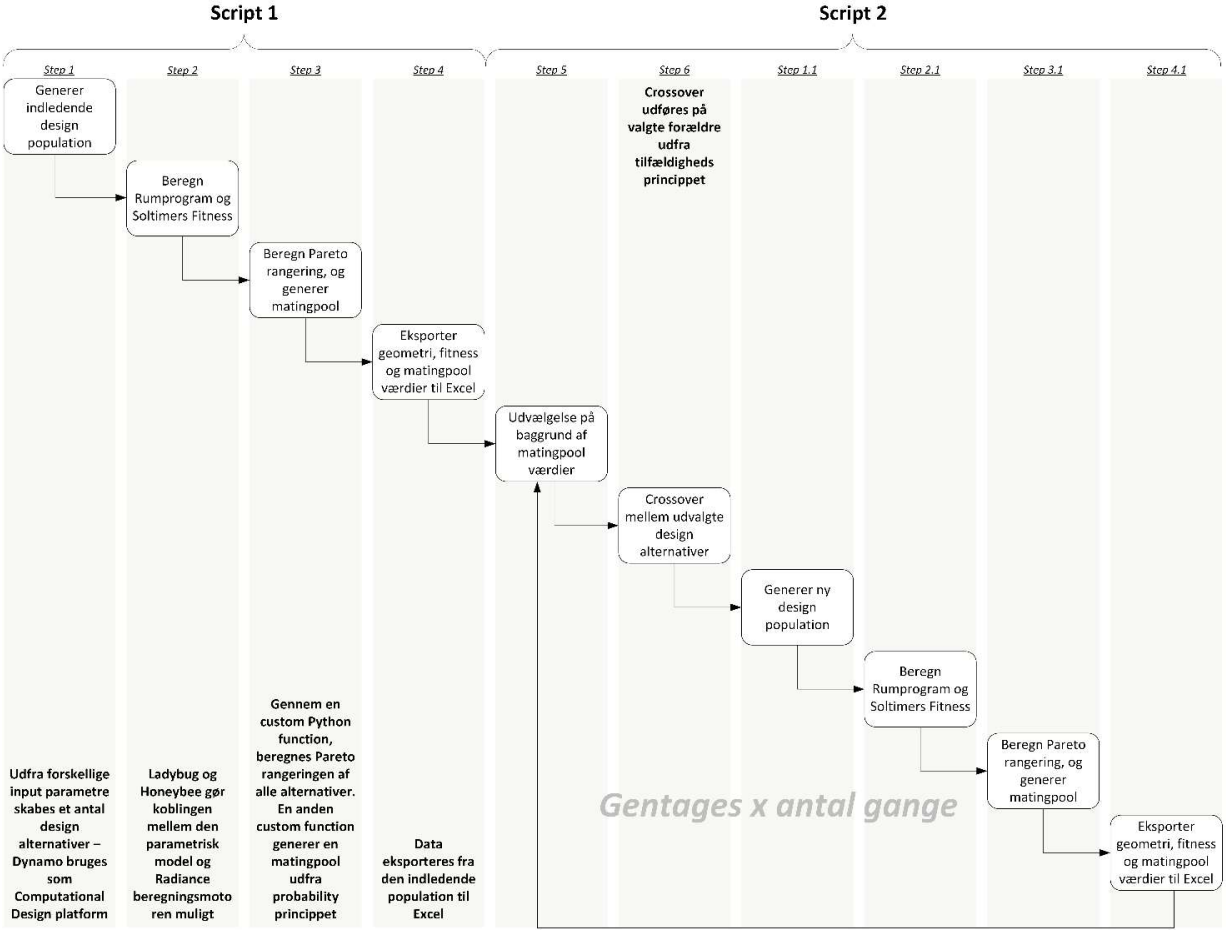
På baggrund af de udforskede elementer inden for både iterativ design og performancebaseret design, er en løsning udviklet, der forsøger at integrere iterativt og performancebaseret design for bedre at kunne udforske et design space og skabe et bedre beslutningsgrundlag i de tidlige faser. Løsningen tager udgangspunkt i det beskrevne testprojekt, men skal også ses som en ramme, der kan benyttes i andre projekter. Løsningen bygges op omkring Multi Diciplinary Optimization (MDO)-metoden og inkorporer to essentielle dele af metoden, parametrisk design og Multi Objective Optimization (MOO). Løsningen i forhold til casen kigger på design spaceudforskning, minimering af skyggepåvirkning på eksisterende forhold og maksimal udnyttelse af rumprogram.

Som tidligere beskrevet består design processen af et ny byggeri af flere forskellige ”konkurrerende” objectives. Et objective skal forstås som et mål, der er vigtigt for det endelige design og derved bliver en vigtig del af designprocessen. Objectives skal være målbare numeriske værdier, hvor målet enten er at maksimere eller minimere værdien. Ofte ses det at der kun arbejdes med ’Single Objectives’ når design analyseres, hvilket gør det problematisk at optimere det endelige design, da objectives i byggeri på mange måder er konkurrerende forstået på den måde at optimering af et objective, kan have negativ indflydelse på andre objectives. I dette løsningsforslag kigges der på to objectives, som er udvalgt på baggrund af bygherres ønsker fra casen, minimering af skyggepåvirkning på eksisterende forhold og opfyldelse af arealkravet. Disse objectives er retvisende for, hvorfor MOO er vigtigt, især når det gælder bygningsdesign. De to objectives kan klassificeres som konkurrerende, minimering af skygger på eksisternde forhold vil logisk optimeres bedst ved at gøre bygningen mindre, mens opfyldelse af arealkravet vil lide under denne optimering. Dette er et godt eksempel på, hvordan objectives ofte er konkurrerende, og dermed gør single objective optimering problematisk og MOO mere passende, og det handler altså om at finde det bedste kompromis.

Det udarbejdet løsningsforslag bygger på seks primære steps og fire sekundære steps, der er gentagelser af de første fire primære steps (se Figur 11-1). Løsningsforslaget bygger som tidligere beskrevet på MDO-metoden og integrerer derved parametrisk design og MOO i en generativ designproces. En GA tilgang er gjort for at kunne udforske det samlede design space på den mest effektive måde ved hele tiden at udvikle de bedste designløsninger. Workflowet er opdelt i to scripts, script 1 køres én

gang som det første led i processen, mens script 2 er opsat således, at det køres i et loop et ønsket antal gange.

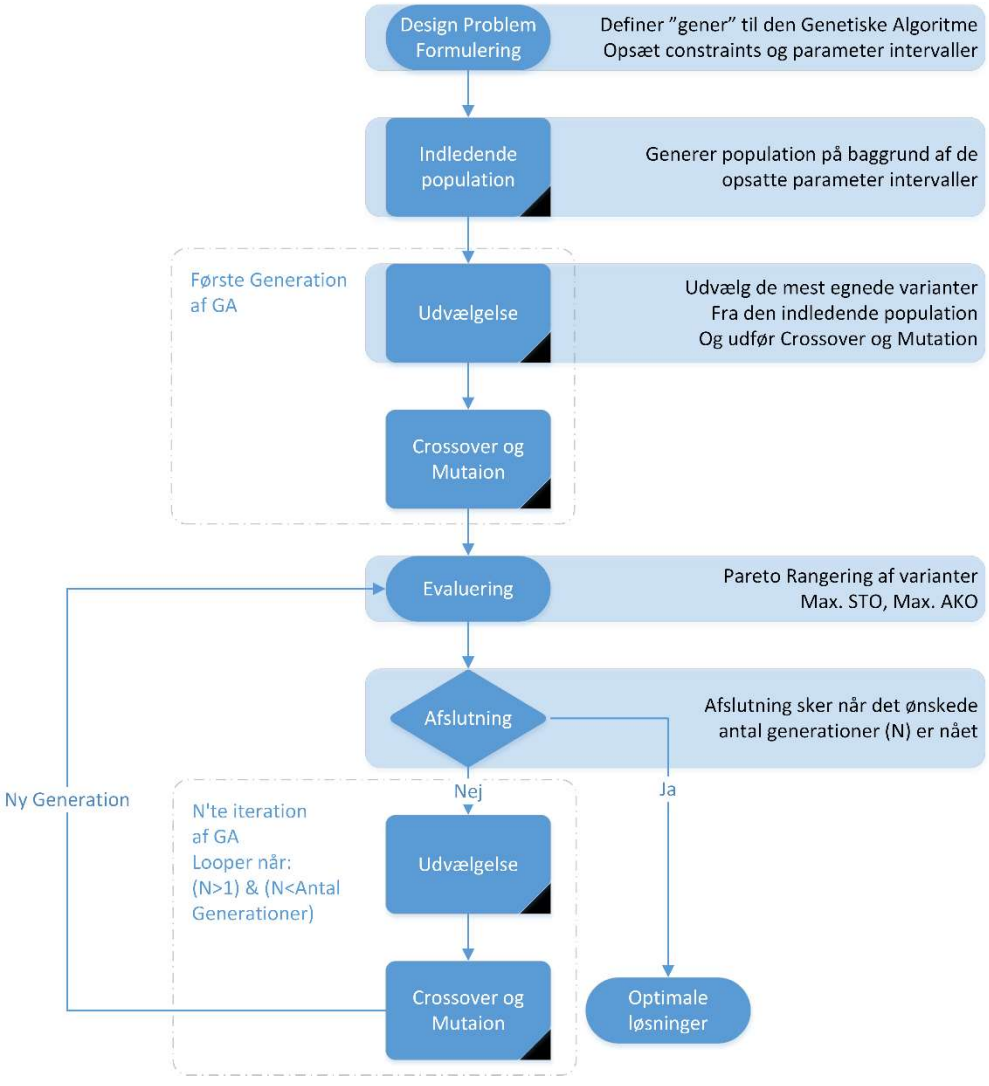
Det første step i processen handler om at genere den indledende population af designalternativer, hvilket sker ved at definere en række parameterintervaller og en populationsstørrelse. Grundet brugen af et VPS miljø til at skabe det parametriske design genereres hele den indledende population af designs automatisk og kræver kun, at brugeren definerer nogle rammer og intervaller for de enkelte parametre. Samtidig med der genereres en population, evalueres alle designvariationer i forhold til de opsatte objectives i step 2. Designet analyseres i forhold til, hvor mange timer solen skinner på en given overflade, og hvor godt designet overholder kravet om antal m2. Ladybug Tools står for sollys analysen, mens et Excel ark med rumspecifikationer sammenlignes med designvarianten for at tjekke, hvor godt varianten møder de opsatte m² krav. For at kunne sammenligne de forskellige designvariationer, bregnes hver variations Pareto-rangering i step 3, hvor alle varianter som udgangspunkt tildeles rangeringen 1, og efterfølgende tilføjes 1 til den rangering hver gang en anden variant dominerer varianten. Udover at rangere hvert alternativ genererer step 3 også en matingpool, som bruges i GA processen, hvorfra de mest egnede designvarianter udvælges. Sidste step i det første script er at eksportere henholdsvis den generede geometri i .obj format samt det relevante data, der skal bruges for at udvikle generationen. Dette skaber grundlaget til de efterfølgende generationer og gør det muligt at tjekke data fra generation til generation samt visualisere hver variation via 3D .obj modellen



Figur 11-1 Løsningsforslagets tekniske proces

Script 2 består af to nye primære steps og gentagelser af de fire steps fra script 1. Script 2 fungerer som et loop, der kører et givet antal gange, før det stopper, og det er op til brugeren at bestemme hvor mange iterationer, loopet skal køre. Hvis ikke et tilfredsstillende resultat er nået efter det angivet antal iterationer, kan processen genoptages med et nyt antal iterationer. Det første step i script 2, eller step fem i den samlede proces, er udvælgelse af de designvariationer, der skal videreføre udviklingen af den næste generation. Udvælgelsen sker på baggrund af den matingpool, der i script 1 blev oprettet og benytter *probability* metoden til udvælge de mest egnede variationer. Step 6 er det sidste primære step i processen, og dette er steppet, hvor data til den nye generation genereres via en crossover og mutationsfunktion.

For at kunne håndtere casens MOO problem, benyttes en genetisk algoritme (se Figur 11-2), hvilket gør det muligt at finde optimale løsninger hurtigere end ved en *'brute force'* løsning. Den genetiske algoritme bygger på principperne om naturlig udvælgelse og Drawins evolutionsteori. Ved at benytte en GA, foretages designoptimering samtidig med, der hele tiden genereres nye og bedre tilpassede løsninger. Formålet med processen er altså ikke at finde ét optimalt design, hvilket som regel ikke er muligt, men i stedet skabes et design space med løsninger, hvor velinformerede afvejningsbeslutninger kan træffes.



Figur 11-2 Løsningsforslagets Genetisk Algoritme

11.1 OBJECTIVES FUNKTIONER

Til casen er der opsat to objectives som skal optimeres, henholdsvis soltimer på eksisterende bygning og overensstemmelse med arealkravet opsat af bygherren. Soltimer på eksisterende bygning (STO) kravet er opsat for at sikre, så meget sollys som muligt rammer de nuværende solceller placeret på taget af den eksisterende bygningen placeret overfor den nye. STO-objectivet skal maksimeres, altså jo flere soltimer jo bedre. Det andet objective, overensstemmelse af arealkrav (AKO), evaluerer, hvor godt et givet design møder de opsatte arealkrav. De to objective funktioner kan udtrykkes ved hjælp af følgende formler:

S_{obj} = Max. STO

A_{obj} = Max. AKO

Hvor,

STO = Soltimer på eksisterende bygning Objective funktion

AKO = Arealkrav overholdelse Objective funktion

11.1.1 SOLTIMER PÅ EKSISTERENDE BYGNING OBJECTIVE FUNKTION (STO)

For at kunne beregne antallet af soltimer på den eksisterende bygning er en række inputs nødvendige. Alle disse inputs skal integreres i det samlede workflow for at kunne benytte resultatet fra hver variation til at udvikle næste generation. Nedenstående tabel viser de nødvendige inputs samt det format, de importeres i og værktøjet, der bruges til at importere.

Inputs	Format	Import Værktøj
Eksisterende omgivelser	.osm	Elk package (Dynamo)
Vejr data	.epw	Ladybug
Analyse motor	Radiance	Honeybee

Tabel 11-1 Oversigt over nødvendig data i STO funktionen

Første nødvendige input i STO-funktionen er de eksisterende omgivelser. For at kunne beregne antal soltimer skal de eksisterende bygningers data importeres til den parametriske model og omdannes til geometri. Dette gøres ved hjælp af det åbne format OpenStreetMap, og opensource Dynamo packagen *Elk*. Gennem openstreetmap.org eksporteres .osm-filen, der indeholder data omkring det eksporterede område. Ved hjælp af Dynamo og Elk packagen importeres data fra .osm-filen til Dynamo, og filens data kan efterfølgende bruges til at genere de eksisterende bygningers geometri. For at kunne beregne det nye designs indvirkning på den eksisterende bygnings adgang til sollys, skal vejrdata fra området importeres. Dette sker ved hjælp af EnergyPlus-filer (.epw), som indeholder vejrdata og bruges af en lang række simuleringsprogrammer. Ved hjælp af Ladybug til Dynamo kan .epw importeres som en del af den parametriske model, hvilket gør det muligt at køre simuleringer. Inden simuleringen kan køres skal en simuleringsmotor importeres, for at beregne antal soltimer anvendes Radiance (Radsite 2017) simuleringsmotoren. Radiance indeholder en række forskellige måder at foretage lyssimuleringer på. For at kunne integrere Radiance-simuleringerne benyttes Honeybee til Dynamo.

11.1.2 AREALKRAV OVERHOLDELSE OBJECTIVE FUNKTION (AKO)

Arealkrav er normalt det første kriterium, som overvejes i forbindelse med et design af et nyt projekt – det er typisk en drivfaktor tidligt i designprocessen. Selvom arealkravets fleksible karakter gør det muligt at gå på kompromis med henblik på at opfylde andre design objectives såsom maksimering af sollys på eksisterende bygning, bør dette objective indgå i enhver form for design space udforskningsproces, der involverer analyser. Input til AKO-beregningen er afledt af to kilder: I) projektkrav fra bygherren og II) modelinformation fra det genererede designalternativ.

Projektets krav importeres til workflowet gennem et Excel plugin i Dynamo, hvor det sammenlignes med den aktuelle designvariations samlede etage m^2. Hvert designalternativ opsplittes per etage, hvorefter etagens samlede m^2 beregnes, hver etageareal sammenlægges til et samlet areal, som til sidst sammenlignes med bygherrekravet. Jo tættere designvariationens samlede areal er på det samlede krav, jo højere en score får variationen i AKO funktionen. Dette sker automatisk, hver gang en ny variation udvikles, hvilket gør det nemmere at håndtere udforskningen af et langt større design space og derved sikre et mere informativt design valg.

11.2 DESIGN PROBLEMFORMULERING

For at kunne anvende en GA til et problem, er der et behov for en matematisk definition af problemet. Når forbindelsen mellem et problem og dets respektive løsningsrum er defineret, kan den systematiske udforskning og evaluering af løsningsrummet opnås. De parametre, der gælder for denne proces, kan opdeles i to primære kategorier; parametre der besidder et interval af acceptable værdier, som definerer løsningsrummet og parametre, der besidder givne værdier, som anvendes til at måle og evaluere variationer. Parametre med et interval af acceptable værdier, der bruges til at definere løsningsrummet, er i forbindelse med denne løsning designparametre. Designparametre er parametre, der styrer de geometriske konfigurationer, og de er designspecifikke og defineres af brugeren. Designparametrene er beskrevet mere detaljeret i følgende afsnit. Parametre med en given værdi bruges til at måle og evaluere præstationen af individuelle designvariationer i henhold til løsningens definerede objective funktioner. Disse parametre kan yderligere opdeles i AKO-parametre og STO-parametre som tidligere beskrevet. Disse værdier fastlægges ud fra en given formel under hver designgenerering i en GA-runde. Desuden kan denne kategori betragtes som genotypeparametre, da de bruges til at definere de "gener", der senere anvendes af løsningens tilpassede GA.

11.2.1 DESIGN GEOMETRI PARAMETRE

Design geometriparametre definerer den geometriske udformning af designet. Som beskrevet i afsnit 8.4.2 er der tre parametergrupper; de drivende, drevne og faste parametre. Løsningen består af fem drivende parametre, fire drevne parametre og tre faste parametre. Udover de i alt 12 parametre består selve geometridannelsen af en lang række logisk opbyggede afhængigheder og beregninger, der tilsammen danner den geometriske model.

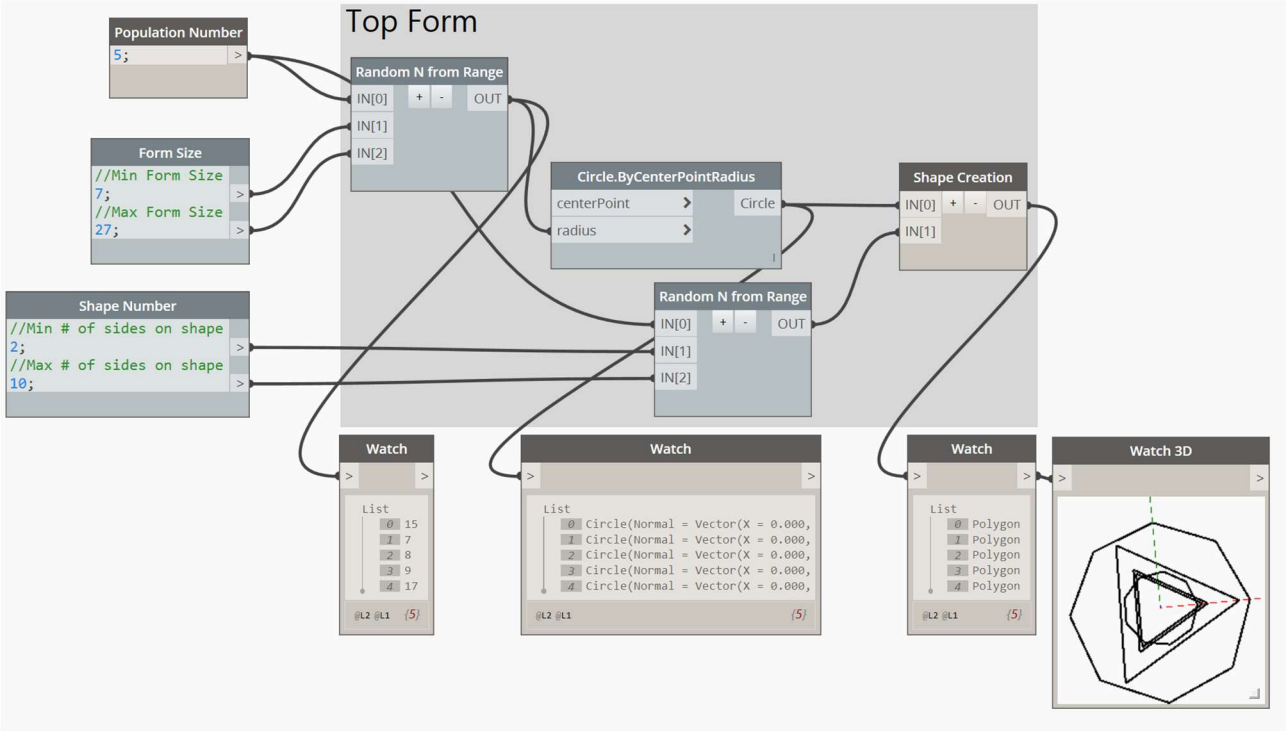
De drivende parametre er alle de parametre, der indeholder et givent interval, der defineres af brugeren, hvor intervallet defineres af en minimum og maksimum værdi for parameteret. *Formstørrelse*parameteret består af to integers, der tilsammen beskriver den mindste størrelse, som formen kan have samt den største størrelse. Dette parameter sørger for, at uinteressante variationer undgås, altså variationer der aldrig vil kunne overholde arealkravet, og sørger for, at sammensætningen af forme vil kunne give et acceptabelt resultat. Antal sider på form definerer en del af logikken bag top og bund formgenereringen, hvor det laveste acceptable antal sider er to, hvilket svarer til en cirkel, mens det højeste er ti, hvilket generer en ti-kant. Formrotation en et interval mellem 0 og 90 grader, og nummeret repræsenterer antallet af grader, den øverste form skal roteres i forhold til XY planet. Top tilt roterer ligeledes den øverste form, men i stedet for XY planet roterer den i forhold til YZ planet.

Drivende Parametre		
Parameter	Interval	Output Datatype
Formstørrelse	minStørrelse..maxStørrelse	Integer
Antal sider på form	2..10	Integer
Formrotation	0..90	Floatingpoint
Top tilt	-15..15	Floatingpoint
Drevet Parametre		
Parameter	Afhængigheder	Output Datatype
Antal etager	Bygningshøjde/Etagehøjde	Integer
Topform	Antal sider på form	Polygon/Cirkel
Bundform	Antal sider på form	Polygon/Cirkel
Faste Parametre		
Parameter	Værdi	Output Datatype
Population størrelse	30	Integer
Bygningshøjde	100	Integer
Etagehøjde	5	Integer

Tabel 11-2 Oversigt og løsningens Design Geometriparametre

Selve formgenereringen består i sig selv af tre primære funktioner, topformgenerering, bundformgenerering og en loftingfunktion, der generer geometri mellem de to former. Top- og bundformgenereringen benytter de to drivende parametre formstørrelse og antal sider på form. Formstørrelsesintervallet kobles sammen med en custom Python

funktion (Figur 11-4), som ud fra den givne max. og min. vælger et tilfældigt nummer, hvilket sker gennem Pythons standard modul *random*. Udover max/min intervallet kobles populationsnummeret også til denne funktion, og gennem det kodede *for loop* generes et tilfældigt nummer mellem max og min værdien et bestemt antal gange. For at genere den endelig top- og bundform bruges endnu en custom Python funktion (Figur 11-5). *Shape Creation* funktionen kræver to inputs; en cirkel og en integer, der repræsenterer formens sideantal (se Figur 11-3). Ud fra disse input genereres enten en polygon eller en cirkel, om det er en cirkel eller en polygon, der genereres bestemmes ud fra kodens *if/else* statement. Er sideantallet mindre end tre laves en cirkel, ellers laves en polygon med det antal sider, som angivet af inputtet. Outputtet af funktionen er som vist i Figur 11-3's "Watch 3D" node en række former.



Figur 11-3 Top/Bundform genereringsfunktion

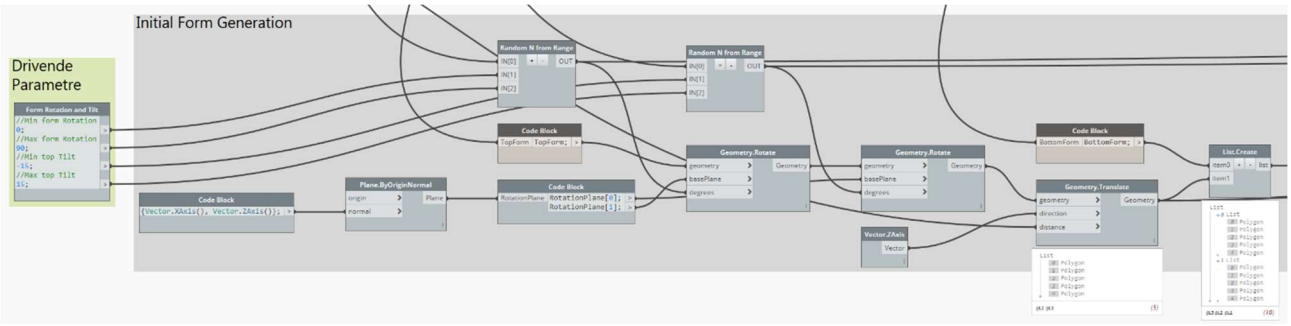
```python
import sys

pyt_path = r'C:\Program Files (x86)\IronPython 2.7\Lib'
sys.path.append(pyt_path)

from random import randint

numOfIterations = IN[0]
minValue = IN[1]
maxValue = IN[2]

randomSequence = []

for i in range(0, int(numOfIterations)):
    randomSequence.append(randint(minValue, maxValue))

#Assign your output to the OUT variable.
OUT = randomSequence
```

Figur 11-4 Custom Python funktion "Random N from Range"

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

startCircle = IN[0]
numberSides = IN[1]

newShape = []

for c, n in zip(startCircle, numberSides):
    if n < 3:
        newShape.append(c)
    else:
        newShape.append(Polygon.RegularPolygon(c, n))

#Assign your output to the OUT variable.
OUT = newShape
```

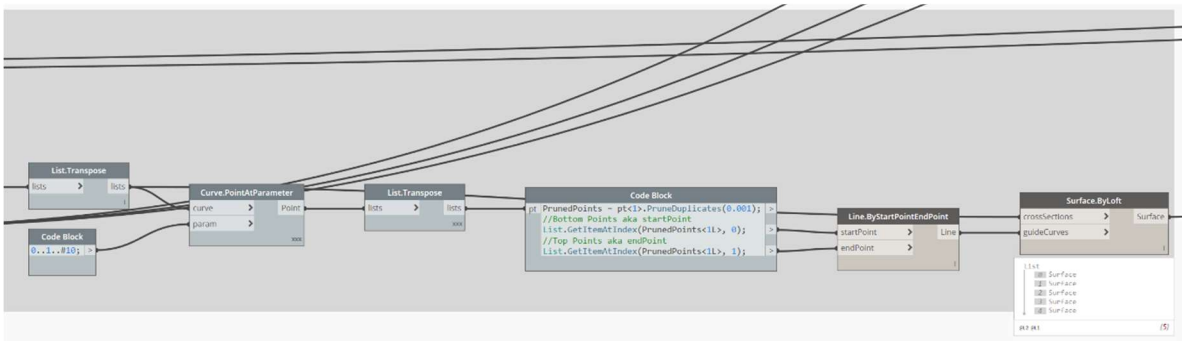
Figur 11-5 Custom Python funktion "Shape Creation"

Efter genereringen af top- og bundformene udføres henholdsvis XY og YZ planrotation på top formen. For at opnå denne rotation, indføres de to sidste drivende parametre Formrotation og Top tilt. Første rotation sker omkring XY planet og benytter igen *"Random N from Range"*-funktionen til at generere et tilfældigt rotations nummer mellem min og max rotationsintervallet. Efterfølgende roteres topformene yderligere denne gang omkring YZ planet for at give formen en tilt funktion. Alle topforme transformeres op til den rigtige højde, og her kommer det faste parameter Bygningshøjde i spil som inputtet til "Geometry.Translate" noden, der sikrer, at topformen er i den rigtige højde. Til sidst samles bund- og topformen i en liste.

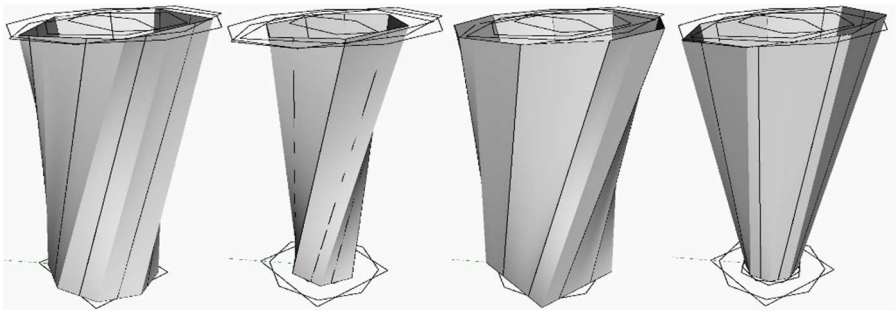


Figur 11-6 Initial Form Generation workflow del 1

Når top- og bundformene er roteret og transformeret, kan den indledende geometri skabes, hvilket gøres ved at *'lofte'* de to forme sammen. Loftningen sker ved at koble top- og bundformlisten sammen med "Surface.ByLoft" noden, som ligeledes kræver en liste "guideCurves", som er linjer, som geometrien former sig efter. Outputtet af at lofte de to former sammen er geometriske former repræsenteret som surfaces (Figur 11-8).



Figur 11-7 Initial Form Generation workflow del 2



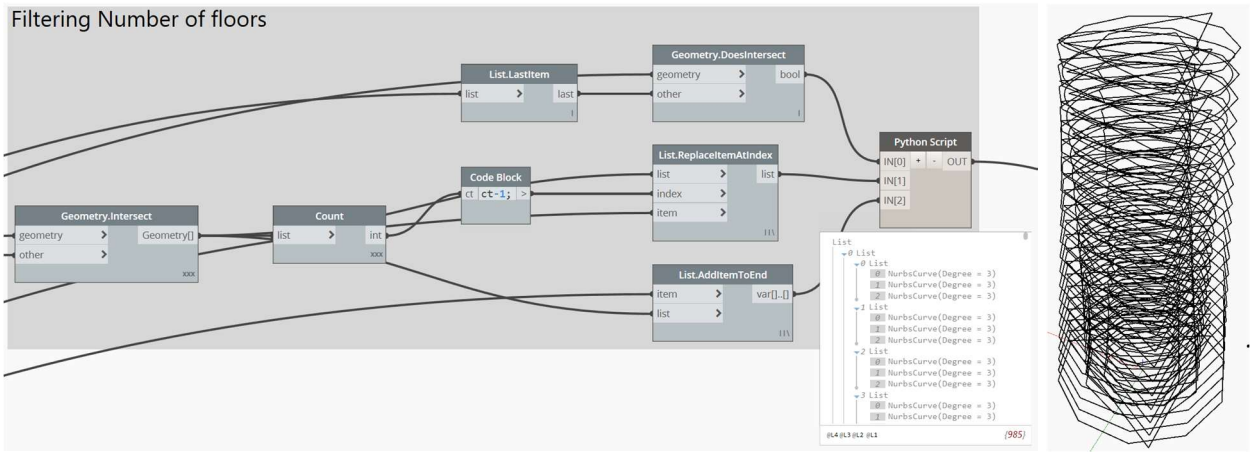
Figur 11-8 Designvariationer efter indledende loftning

For senere i workflowet at kunne evaluere et designs overensstemmelse med arealkravet er det nødvendigt at opdele hvert design i etager. Efter den indledende formgenerering (Figur 11-6 & Figur 11-7) er hele designet modelleret som et samlet surface, hvilket gør det svært at beregne det samlede etageareal. At opdele designet i etager består af to overordnede skridt, I) find antal etager og opret planes ved alle etager, II) opdel geometrien ved hvert skæringspunkt med et etage plane.

Antallet af etager er et drevet parameter, der findes ved at dividere de to drivende parametre Bygningshøjde og Etagehøjde med hinanden. For at lave planes for hver etage er det nødvendigt at finde alle etagers oprindelsespunkt, som bestemmer højden af hver etage. Når oprindelsespunkterne er fundet, omdannes disse til planes.

Figur 11-9 Oprettelse af planes ved alle etager

Skridt to opdeler bygningsformen i etager ved finde de netop oprettede planes skæringspunkter med designvariationens geometri. Outputtet af denne del er en liste af *NurbsCurves,* der repræsenterer hver etages outline (se Figur 11-10).



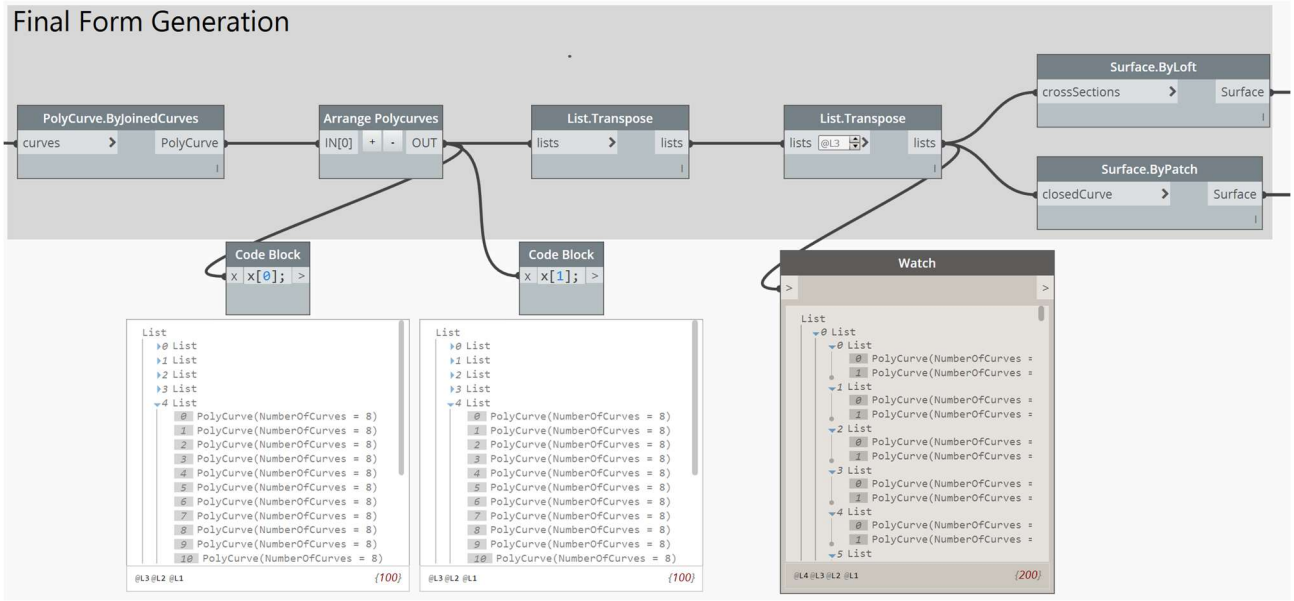
Figur 11-10 Opdel design variationerne i etager ved at finde skæringspunktet mellem geometrien og etage planes

I sidste del af formgenereringsworkflowet genereres den endelige form opdelt på etager. Dette sker ved at omdanne alle *NurbsCurves* til *PolyCurves* og derefter *lofte* dem sammen, hvilket skaber skallen af formen. Udover skallen genereres der også etagegulvflader, som bruges senere til at finde formens samlede etage m². For at sikre at formen opdeles på etageniveau, er det nødvendigt at arrangere de genererede PolyCurves, sådan at de opdeles i lister af en bund PolyCurve og en top PolyCurve. Ligeledes skal de enkelte PolyCurves gå igen flere gange, sådan at top PolyCurven i den ene liste bliver bund PolyCurve i den næste liste. For at opnå dette har det været

nødvendigt at programmere en special funktion i python (Figur 11-11). Funktionen kræver et input af PolyCurves gennem et for loop arrangere funktionen alle PolyCurves i enten en start curve-liste, eller end curve-liste. Outputtet af funktionen er de to curve-lister, henholdsvis start og end-curve.

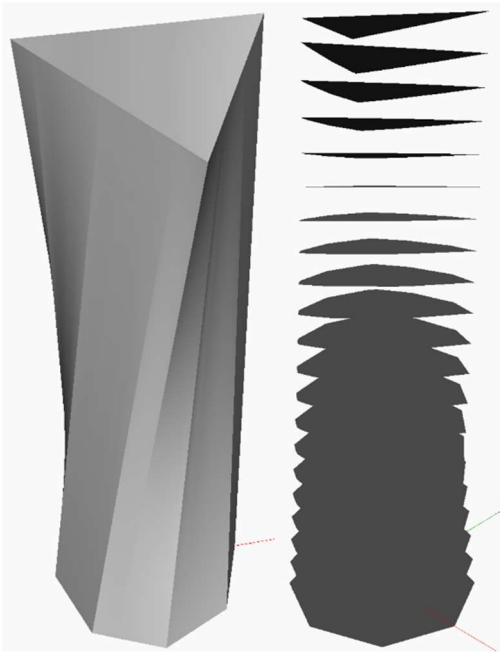
```
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

polyCurves = IN[0]

ListOfstartCurves = []
ListOfendCurves = []

for polyCurveList in polyCurves:
    startCurve = []
    endCurve = []
    ListOfstartCurves.append(startCurve)
    ListOfendCurves.append(endCurve)
    for crv in polyCurveList:
        if len(startCurve) == 0:
            startCurve.append(crv)

        elif len(startCurve) == len(polyCurveList) - 1:
            endCurve.append(crv)

        else:
            startCurve.append(crv)
            endCurve.append(crv)

    for start, end in zip(startCurve, endCurve):
        if start.StartPoint.Z == end.StartPoint.Z:
            startCurve.pop(startCurve.index(start))
            endCurve.pop(endCurve.index(end))

# Assign your output to the OUT variable.
OUT = ListOfstartCurves, ListOfendCurves
```

Figur 11-11 Arrange PolyCurves custom funktion

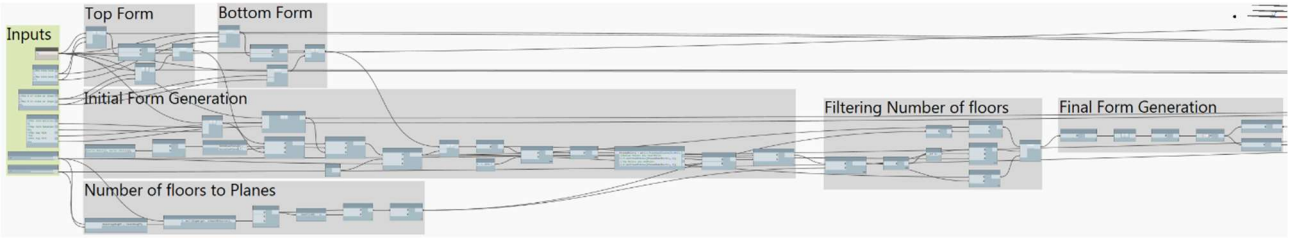


Figur 11-12 Workflow over generering af den endelige form

Når PolyCurves er blevet arrangeret i to lister, lægges de to lister sammen ved at tage et element fra hver liste ad gangen og lægge dem sammen i en ny liste. Dette giver en liste med en række sublister, der hver især indeholder to PolyCurves; top og bund. Ved at lofte mellem disse PolyCurves genereres skallen igen opdelt på etageniveau. For at generere gulvfladerne bruges de samme PoluCurves, men i stedet for at Lofte dem genereres et surface ud fra den lukkede PolyCurve.



Figur 11-13 Endelig form, til venstre skallen, til højre etage gulve



Figur 11-14 Oversigt over det samlede form genererings workflow

11.3 EVALUERINGSPARAMETRE

Evalueringsparametrene er de parametre, der har en direkte indflydelse på nye generationers design geometriparametre. Disse parametre bruges til at evaluere et design ud fra de objectives, der er opsat på projektet. I denne løsning er der på baggrund af casen opsat to evalueringsobjectives, AKO og STO. Disse parametre er derfor styrende i udviklingen af nye designalternativer. Evalueringsparametrene er GA'ens genotypeparametre, da de bruges til at forme løsningens udtryk.

For at kunne evaluere hver designvariation, skal alle evalueringsparametre integreres i det samlede workflow. Ved hjælp af forskellige værktøjer kan alle parametre evalueres i det samme miljø. En beskrivelse af workflowet bag AKO evaluering og STO evaluering vil blive givet i følgende afsnit.

11.3.1 EVALUERING AF AKO OBJECTIVE

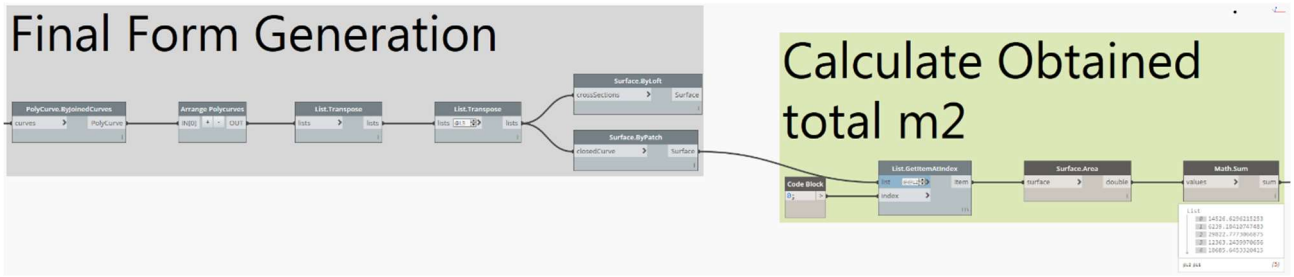
For at kunne evaluere et design i forhold til hvor godt det overholder kravet om areal, skal designets samlede areal beregnes, hvorefter det sammenlignes med projektets arealprogram. Arealprogrammet er udleveret i Excel format (Figur 11-15), hvilket gør det muligt at indlæse dets data direkte i Dynamo, hvorefter alle arealkrav kan lægges sammen for at finde det samlede krav (Figur 11-16).

s/N	Rooms and Facilities	Qty	Approx.Area (m2)	Total (m2)	Department
1	Universal Design Lab	1	240	240	1
2	Centre for Vertical Transportation	1	1000	1000	1
3	Centre for Coastal Protection AR/VR Lab	1	180	180	1
4	Centre for Coastal Protection Data Centre & Risk Mapping	1	120	120	1
5	Quality Learning Lab	1	300	300	1
6	National Geological Centre	1	200	200	1
7	Rock Core Storage	1	600	600	1
8	Construction Productivity Lab	1	800	800	1
9	Mock-up (Retail)	1	81	81	2
10	Mock-up (Healthcare)	1	81	81	2
11	Mock-Up Unit (Hotel)	1	81	81	2
12	Mock-Up Unit (Classroom)	1	90	90	2
13	Mock-Up Unit (Residential)	1	81	81	2
14	Mock-Up Control Room & Store	1	50	50	2
15	Research Lab (Air-conditioning)	1	200	200	2
16	Research Lab (Design and Simulation)	1	200	200	2
17	Research Lab (Lighting Technology)	1	200	200	2
18	Research Lab (IEQ)	1	160	160	2
19	BERII Resource Centre	1	160	160	2
20	Experiential Learning Hub	1	600	600	2
21	National Building Energy Efficiency Centre	1	250	250	2
22	IT Lab	8	120	960	2
23	Design Studio	6	120	720	2
24	VDC Lab	6	120	720	2
25	E-Learning Production Studio	1	160	160	2
26	Teaching & Learning Lab	4	200	800	2
27	Cave Lab	2	68	136	2
28	AR/VR Lab	1	112	112	2
29	Lecture Theatre (300 pax)	2	360	720	2
30	Lecture Theatre (200 pax)	3	200	600	2
31	Multipurpose Hall cum Exhibition Space (Large)	1	800	800	2
32	Multipurpose Hall cum Exhibition Space (Small)	1	400	400	2
33	Library	1	1000	1000	2
34	Office Space	1	2200	2200	3
35	Board Room (33 pax at main table)	1	140	140	3
36	Meeting Room (Large) (33 pax at main table)	3	100	300	3
37	Meeting Room (Medium) (15 pax at main table)	5	60	300	3
38	Meeting Room (Small) (9 pax at main table)	4	30	120	3

Figur 11-15 Casens arealprogram, oprettet i Excel

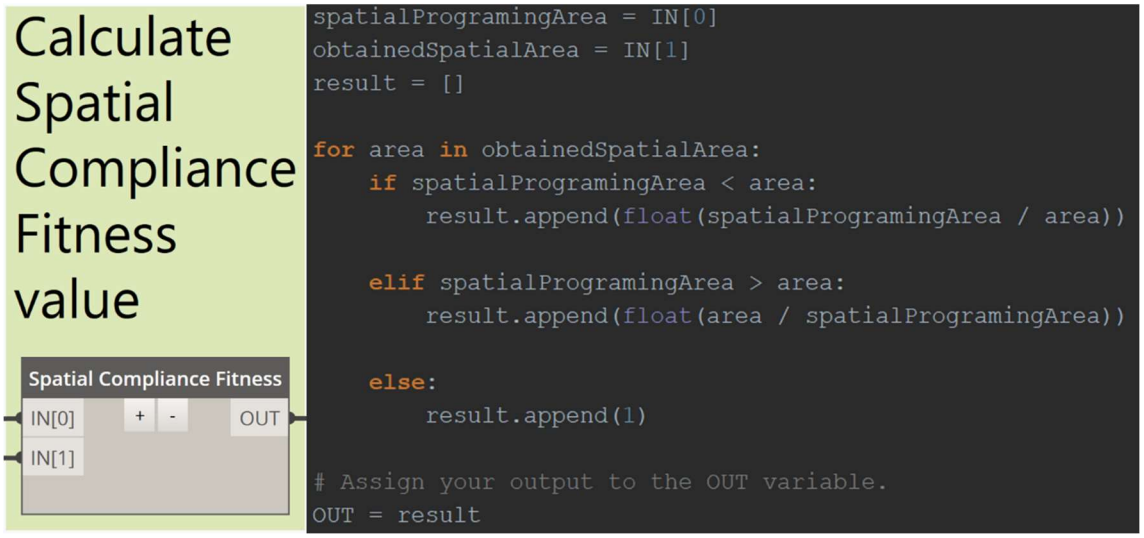
Figur 11-16 Indlæs Excel data, og udregn samlede areal krav

For at beregne designvariationens samlede areal benyttes de generede etagesurfaces, og ud fra en liste med disse surfaces beregnes hver etages areal. Når arealet på alle etager er hentet, summeres de alle sammen til et samlet areal, som derefter kan sammenlignes med arealkravet.



Figur 11-17 Beregn samlede areal for designvariation

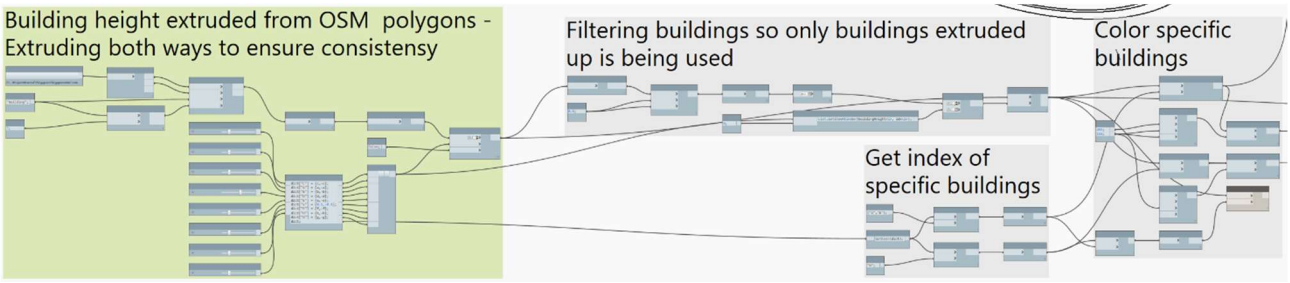
Når det samlede arealkrav er importeret i workflowet, og designalternativernes samlede areal er beregnet, kan de to sammenlignes for at vurdere de enkelte designs fitness. Til at lave sammenligningen er udarbejdet en special fitnessfunktion. Fitnessfunktionen normaliserer forskellen mellem arealkravet og det opnåede designareal, og den vil altid give et tal mellem 0 og 1, hvor målet er at komme så tæt på 1 som muligt. Funktion tjekker de to arealer efter, hvilket der er størst, og hvis det opnåede designareal er større end kravet, divideres kravet med det opnåede areal. Hvis kravet er større end det opnåede, divideres kravet med det opnåede areal. Hvis hverken kravet eller det opnåede areal er størst, betyder det, at de to arealer er ens, hvilket betyder en fitness på 1.



Figur 11-18 AKO fitness function

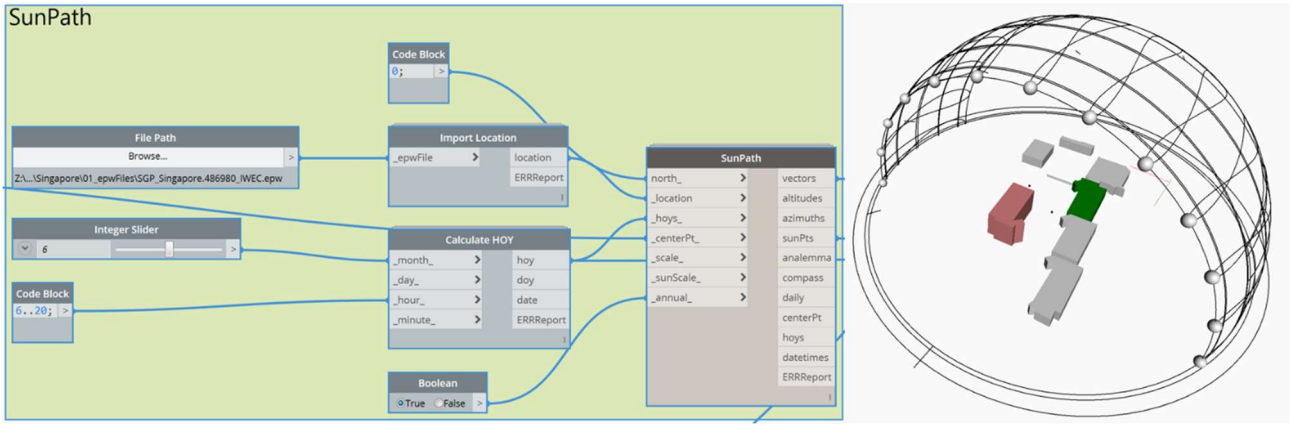
11.3.2 EVALUERING AF STO OBJECTIVE

STO parameteret kræver en del mere beregning og generering og dermed også mere computerkraft. For at beregne antallet af soltimer på eksisterende bygning er der mange faktorer, der kommer i spil. Først og fremmest skal de eksisterende omgivelser importeres ind i den parametriske model (se Figur 11-19), hvilket sker ved hjælp af OpenStreetMap formatet, og Dynamo packagen Elk. Data fra OSM formatet bruges til at genere geometri, der repræsenterer de eksisterende omgivelser. Når geometrien er genereret, filtreres den bygning, der er på området, hvor den nye bygning skal placeres væk. Derudover identificeres bygningen med solceller, hvorefter overfladen, hvor cellerne er placeret, filtreres ud, så det kan bruges i analysen.



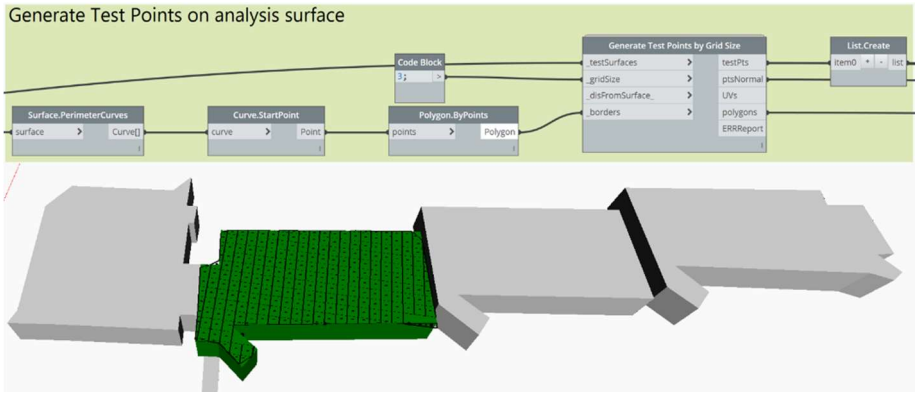
Figur 11-19 Workflow for at importere eksisterende omgivelser gennem OSM formatet

Efter de eksisterende omgivelser er importeret, skal vejrdata fra området ind i workflowet. Denne data kommer fra EnergyPlus Weather-filer. For at kunne loade data fra disse filer i workflowet, bruges Ladybug packagen, der ud fra .epw-filen generer alt den nødvendige data til at udføre soltimeranalysen. Gennem Ladybug har man også mulighed for at bestemme, hvilke måneder, dage og timer man gerne vil lave analyser på.



Figur 11-20 Import af .epw vejr data + Model efter import af omgivelser og vejrdata

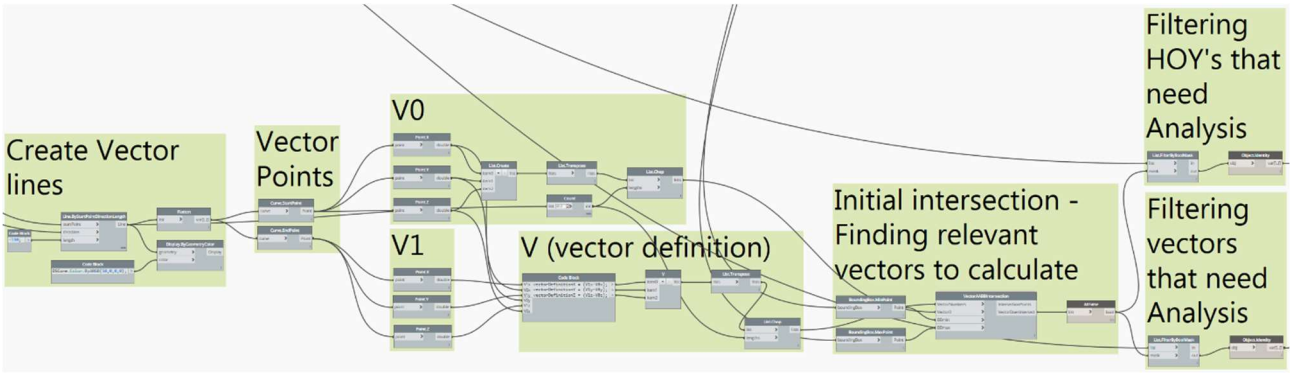
Næste skridt mod soltimeranalysen er at udvælge den overflade, der skal analyseres. For at kunne beregne mængden af soltimer skal der generes testpunkter på overfladen, hvor disse testpunkter bliver grundlaget for selve analysen. Resultatet af selve analysen er et tal, der beskriver, hvor mange soltimer der er på et givet punkt. For at genere testpunkterne bruges Honeybee packagen og noden *"Generate Test Points by Grid Size"*. Denne node kræver tre inputs; overfladen der skal analyseres, gridstørrelse og overfladens grænser.



Figur 11-21 test punkter genereret på analyseoverflade

Analysen, der beregner antal soltimer, bruger de genererede testpunkter og solvektorer, der kommer fra .epw-filen via Ladybug. Jo flere timer der er valgt at analysere, jo flere sol vektorer er der. Hvert testpunkt og solvektorer danner tilsammen en ny vektorer, der bruges i beregningen, og hvis en af disse vektorer blokeres, betyder det, at det givne punkt ikke opnår sollys i solvektorens tid. For at opnå et acceptabelt resultat i analysen er det nødvendigt at genere nok testpunkter, hvor antallet af punkter er afhængigt af overfladens størrelse. Ligeledes vil man gerne tjekke så mange tidspunkter som muligt, hvilket giver flere solvektorer og dermed flere beregninger.

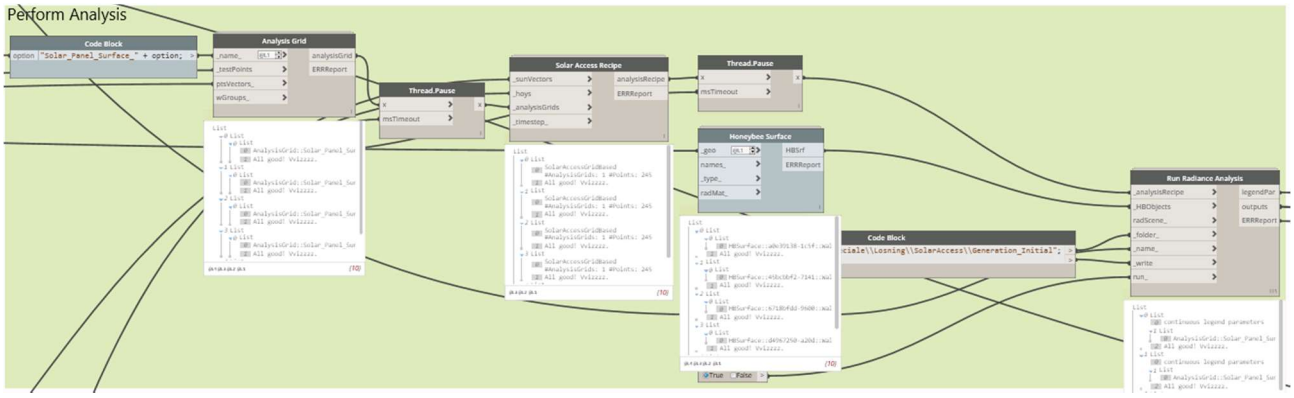
Grundet ovenstående er performance en ekstremt vigtig faktor i dette workflow. For mange beregninger kan føre til crash, mens for få kan føre til et utilstrækkeligt analyseresultat. For at tage hånd om eventuelle performanceproblemer er der til dette workflow lavet et custom Python script, der på en effektiv og performancevenlig måde minimerer antallet af tunge beregninger. Ved at bruge en Vektor/AABB[7] Intersection algoritme kan de vektorer, der er nødvendige at beregne, filtreres ud. Eftersom det kun er konsekvensen af det nye design, der er interessant i denne sammenhæng, er det kun nødvendigt at kigge på de vektorer, der er i karambolage med det nye design. En nærmere beskrivelse af hvordan denne funktion virker kan findes i Bilag 1. Dette kan spare mange beregninger, hvilket i sidste ende vil gøre hele workflowet hurtigere.



Figur 11-22 Vektor filtrering ved hjælp af Vektor/AABB algoritme

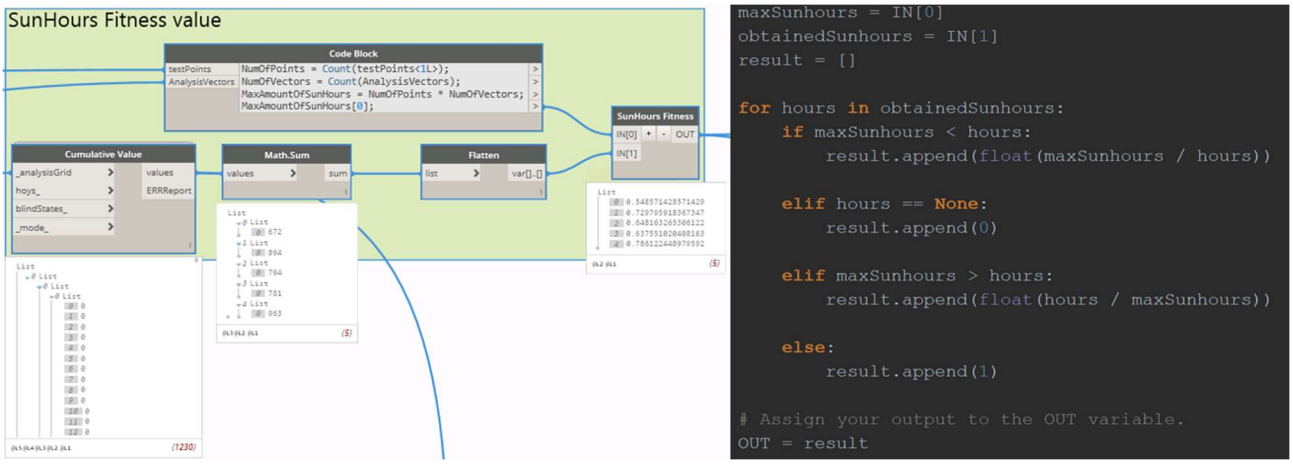
[7] Axis Aligned Boundingbox

Efter alle forberedelserne såsom at genere testpunkter, genere solvektorer og filtrere dem kan den endelige analyse foretages. Langt de fleste af de analyser, der kan foretages i Ladybug Tools, består af minimum tre steps og dertilhørende komponenter. Hvert step og komponent foretager en del beregning der bruges som input i det samlede analysekomponent. Første step i soltimerberegningen er at oprette et analysegrid. Griddet består at de testpunkter, der tidligere er oprettet på den overflade, der skal analyseres, og derudover skal griddet gives et unikt navn. Næste skridt er en såkaldt opskrift til selve analysen. Denne komponent samler testpunkterne og de solvektorer, der skal analyseres og forbereder dem til den samlede analyse. Inden den endelige analyse kan køres, skal den geometri, hvis skyggepåvirkning skal analysere, omdannes fra Dynamogeometri til Honeybeegeometri. Den endelige analyse kan herefter fortages med inputs fra de før beskrevne inputs og et unikt navn til hver analyse. Når "Run Radiance Analysis" Honeybee komponent startes, sender komponenten automatisk den nødvendige data videre til Radiance analysemotoren, som derefter laver de nødvendige beregninger og gemmer de nye data ned i en fil. Honeybeekomponenten læser efterfølgende den gemte data fra Radiance-filen, og outputter det i Dynamo.

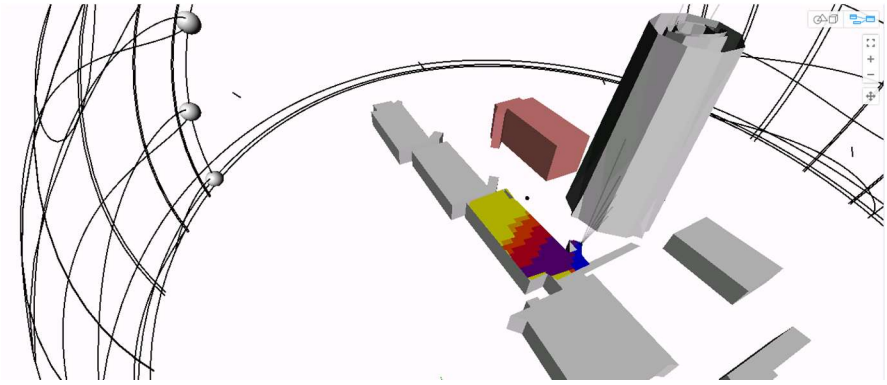


Figur 11-23 Analyseworkflow til soltimesimulering, består af tre indledende Honeybeekomponenter og et afsluttende, der udfører selve analysen

Resultatet af analysen bruges efterfølgende til at udregne designvariationens fitness, så designet kan evalueres. Komponenten "Run Radiance Analysis", som udførte beregningen, outputter data fra den fil, hvor analysen er gemt i. Dataene består af udregninger for hvert testpunkt og indikerer hvor mange vektorer, der rammer de enkelte punkter. Alle tallene fra hver designvariationsanalyse sammenlægges, så hver variation har ét tal, som er den samlede mængde af soltimer på hvert punkt. Den maksimale soltimeværdi udregnes ved at tage antallet af alle testpunkter, og gange med antallet af vektorer. For at finde frem til soltimefitness bruges samme metode som i AKO fitnessfunktionen. Hver soltimeanalyseresultat sammenlignes med den maksimale værdi, hvorefter tallet normaliseres. Jo tætter på 1 jo bedre.



Figur 11-24 Workflow for udregning af fitnessværdien for soltimer objectivet



Figur 11-25 Visualisering af analyseresultatet, farvekodet gul er højeste antal timer mens blå er laveste

11.4 GENETISK ALGORITME

Løsningens genetiske algoritme (GA) anvender et værdikodningssystem, der består af to forskellige værdidatatyper, integreres og floating points. Dette giver som tidligere beskrevet den nødvendige fleksibilitet i forhold til at bruge en parametrisk modelleringsplatform, hvor designparametrene er styret af reelle talintervaller.

Hver designvariation er et kromosom bestående af 10 forskellige gener (Tabel 11-2), der består af et interval af acceptable værdier. Et gen er altså en enkelt værdi, der ligger indenfor det givne interval.

Den genetiske algoritme er bygget op om tre funktioner; population, evaluering og selektion samt crossover og mutation. Hver af disse funktioner skaber tilsammen en udvikling i designet og gør det muligt kun at genere designalternativer med gener, der skaber bedre designs.

11.4.1 POPULATION

Løsningens workflow består af to scripts. Det første script skaber den indledende population, mens det andet script kører i loop, til et ønsket antal generationer opnås. Den første population oprettes tilfældigt, generne i hvert kromosom udvælges tilfældigt og sammensættes, så hvert kromosom danner en designvariant, der kan analyseres. Størrelsen på den første population er afgørende for, hvor godt et design kan udvikles. Variation i population er altafgørende for at opnå en udvikling i designet. Der findes flere svar på, hvor stor en population skal være i en GA, dog er der

ikke et endeligt svar, der altid gør sig gældende. Den mest benyttede tommelfingerregel siger, at population skal være 10x større end antallet af gener. Det vil sige, at hvis hvert kromosom består af 10 gener, skal populationen være 10x10 = 100. Dette er dog kun en tommelfingerregel, og der findes ligeledes forsøg, der peger på, at en for stor population kan være skadende for udviklingen. I afsnit 13 er der lavet forsøg med forskellige populationsstørrelser.

Populationsstørrelsen er brugerbestemt og bestemmes altid i det første script. Den indledende population skabes ud fra de beskrevne parametre i afsnit 11.2.1 og 11.3. Alle kromosomernes genværdier samt deres fitness samles i et Excel ark, så udviklingen i generationerne kan følges.

Name	TopRadius	TopFormNum	BottomRadius	BottomFormNum	Rotation	Tilt	BuildingHeight	FloorHeight	SpatialComplianceFitnessValue	SunHoursFitnessValue	OptionRanking	MatingPool
New_Building_Option1	16	4	13	8	72,014554	3,855074	100	5	0,615100721	0,814285714	1	48
New_Building_Option2	20	5	14	8	89,511772	10,22694	100	5	0,914834374	0,779591837	1	59
New_Building_Option3	20	7	22	8	41,04043	1,694485	100	5	0,645816068	0,731632653	47	93
New_Building_Option4	12	6	19	3	70,448449	11,78838	100	5	0,535914359	0,784183673	27	76
New_Building_Option5	26	3	14	9	71,728601	11,40502	100	5	0,900094237	0,784183673	1	1
New_Building_Option6	8	7	9	10	52,156879	11,69722	100	5	0,258534186	0,864285714	1	12
New_Building_Option7	10	6	22	2	16,165486	11,83048	100	5	0,954080025	0,767857143	7	21
New_Building_Option8	7	7	24	3	2,550345	1,096609	100	5	0,52958359	0,780612245	33	32
New_Building_Option9	13	5	13	5	22,03821	6,615099	100	5	0,49277927	0,823979592	1	46
New_Building_Option10	26	5	8	2	85,059857	5,022249	100	5	0,953785794	0	10	4
New_Building_Option11	10	10	12	9	77,56863	-14,3911	100	5	0,441617843	0,828571429	3	93
New_Building_Option12	26	4	24	3	61,758959	-11,306	100	5	0,78573523	0,697959184	34	59
New_Building_Option13	22	9	11	2	82,413803	0,796426	100	5	0,977621105	0,770918367	1	94
New_Building_Option14	11	7	15	8	87,8408	-1,48856	100	5	0,605528339	0,807653061	5	64
New_Building_Option15	23	10	16	4	27,603997	-14,9054	100	5	0,91017696	0,746428571	15	54
New_Building_Option16	10	7	10	5	47,964063	-12,1824	100	5	0,316146883	0,853061224	1	91
New_Building_Option17	19	10	17	9	74,23701	-2,43057	100	5	0,847294421	0,75255102	22	89
New_Building_Option18	13	3	17	9	15,587924	-3,16342	100	5	0,64722002	0,782142857	16	4
New_Building_Option19	19	10	23	10	59,26257	-11,635	100	5	0,614476748	0,725	58	81
New_Building_Option20	18	6	10	2	88,834345	-7,62975	100	5	0,682225453	0,807142857	3	21
New_Building_Option21	21	5	10	8	88,038155	0,963797	100	5	0,762650351	0,79744898	2	64
New_Building_Option22	17	6	13	2	80,789365	12,52539	100	5	0,787846057	0,790816327	1	86
New_Building_Option23	12	10	23	2	35,329504	-1,02035	100	5	0,788549464	0,753061224	28	59
New_Building_Option24	27	5	22	10	85,212969	0,051131	100	5	0,509513745	0,70255102	77	61
New_Building_Option25	25	10	17	3	25,612243	11,34633	100	5	0,808654729	0,729591837	30	66
New_Building_Option26	14	2	13	3	12,188601	3,717746	100	5	0,502289315	0,81377551	5	61
New_Building_Option27	27	6	25	2	0,5749931	2,341193	100	5	0,410638431	0,669387755	90	5
New_Building_Option28	16	10	20	4	4,2478634	4,630008	100	5	0,981216928	0,766836735	3	86
New_Building_Option29	14	9	20	10	51,085251	11,21435	100	5	0,923384015	0,759693878	10	8
New_Building_Option30	9	3	12	10	35,000852	12,70303	100	5	0,320265695	0,842346939	5	85
New_Building_Option31	19	4	13	4	32,338687	6,790964	100	5	0,606418497	0,783163265	21	91
New_Building_Option32	14	9	17	3	52,000901	3,815839	100	5	0,590928899	0,787244898	20	76
New_Building_Option33	11	4	20	10	76,720674	-12,4857	100	5	0,84656878	0,78622449	1	85
New_Building_Option34	16	8	24	10	15,258363	10,40188	100	5	0,667102302	0,729591837	46	8
New_Building_Option35	15	6	11	9	21,470803	-11,7816	100	5	0,565682139	0,816326531	2	15
New_Building_Option36	19	4	18	2	85,946059	1,402256	100	5	0,903774121	0,764795918	12	58
New_Building_Option37	15	9	15	2	29,063601	0,199024	100	5	0,857599916	0,781122449	3	64
New_Building_Option38	12	4	23	5	56,915963	-8,59009	100	5	0,923294231	0,768877551	5	54
New_Building_Option39	27	4	19	5	62,330242	9,053953	100	5	0,712524936	0,731632653	40	83
New_Building_Option40	15	7	13	5	50,316762	12,84631	100	5	0,619293679	0,807142857	3	94
New_Building_Option41	8	4	19	4	69,694025	1,451884	100	5	0,49591166	0,812755102	7	83
New_Building_Option42	8	9	14	8	34,790755	12,53325	100	5	0,451553325	0,831632653	2	4
New_Building_Option43	15	9	18	4	80,711494	-12,9481	100	5	0,809177554	0,780612245	6	83
New_Building_Option44	10	10	10	9	30,794274	8,318691	100	5	0,36405151	0,843877551	1	94
New_Building_Option45	8	6	18	7	48,097516	-4,12095	100	5	0,622967239	0,8	6	66
New_Building_Option46	8	10	21	5	43,705959	10,18454	100	5	0,73041191	0,796938776	4	88
New_Building_Option47	21	10	7	4	48,253209	5,231802	100	5	0,697578051	0,81377551	1	1
New_Building_Option48	17	4	24	6	20,062772	8,973182	100	5	0,777606753	0,728571429	34	48

Generation_8 | Generation_7 | Generation_6 | Generation_5 | Generation_4 | Generation_3 | Generation_2 | Generation_1 | **Generation_0** | Template

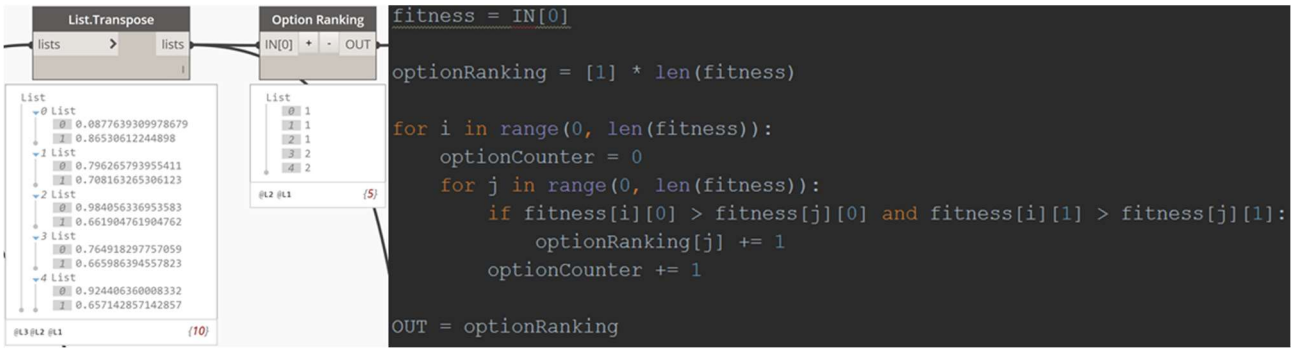
Figur 11-26 Udklip af Excel ark der lagrer alle populationsgenerationers kromosomdata

11.4.2 EVALUERING & SELEKTION

For at udvikle en ny generation i en fit retning, skal de rigtige variationer udvælges til at videregive deres gener til den nye generation. Udvælgelsen af forældre til den nye generation sker via *probability*-metoden på baggrund af de enkelte variationers Pareto-rangering. Dermed er evalueringen af variationerne det første skridt i at udvælge forældre.

Evalueringen af designvariationer laves på baggrund af deres fitnessscore. Hver variant har to fitnessfunktioner, som hver især skal optimeres på baggrund af forskellige parametre. I dette projekt er de to fitnessfunktioner henholdsvis overholdelse af arealkravet og optimering af soltimer på eksisterende bygning. Disse to objectives er i sig selv konkurrerende, hvilket vil sige, at en fuld optimering af begge objectives er næsten umulig. I stedet skal de bedste kompromis findes. Eftersom de bedste kompromis skal findes, kan en simpel sammenlægning af begge fitnessfunktioner ikke bruges, eftersom dette vil udelukke en lang række variationer. I stedet bruges Pareto optimalmetoden, hvor en variation er Pareto optimal, når en forbedring i en fitnessfunktion vil føre til forringelse i en anden. På denne måde skabes et nyt design space, hvor kun de bedste generede alternativer er placeret.

Pareto-metoden bruges som en rangeringsmetode, der sammenligner alle varianter i en generation og rangerer dem efter, hvor mange dominerende variationer en variant har. For at udføre denne rangering tjekkes alle variationers fitnessværdier mod hinanden. Hver variant starter med rangeringen 1, for hver gang en anden variant dominerer varianten, tillægges en til den nuværende rangering. Det vil sige, at en variant, der domineres af én anden variation, får rangeringen 2, en, der domineres af to, får rangeringen 3 og så videre. Selve rangeringsmetoden er lavet som en simpel custom python node, der looper igennem alle fitnessværdier og tilføjer til variationens rangering, hvis en anden variation dominerer den. Den eneste måde, en variation kan dominere end anden, er, hvis den har en bedre score i begge fitnessfunktioner. Det vil sige, at hvis variation A har en fitness score på 0,9 og 0,3 mens variation B har en score på 0,8 og 0,97 er der ingen af disse variationer, der dominerer hinanden. Variation A er bedre på første parameter, mens variation B er bedre på det sidste. På denne måde får brugeren en større frihed til at vælge hvilke fitnessfunktioner, der har størst betydning og hvilken, der bedst kan gøres et kompromis ved.

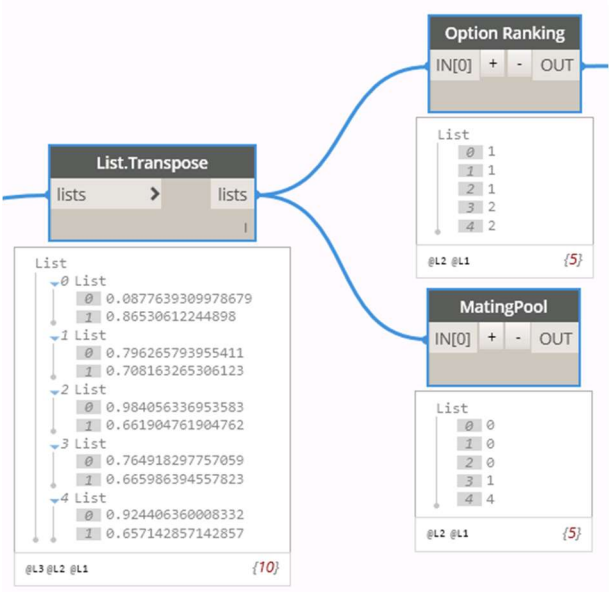


Figur 11-27 Pareto rangeringsmetode, simpelt python loop der tillægger en værdi oven i rangeringen hvis varianten domineres af en anden variant

Udover evaluering af de forskellige varianter skal der udvælges en række forældre fra den gældende generation til at udvikle den nye generation. De forældre, der udvælges, placeres i en pulje, hvorfra de forskellige varianter krydses for at lave nye varianter til den næste generation. Puljen med forældre kaldes en matingpool. Som udgangspunkt findes der to forskellige principper, når der skal laves en matingpool. *Elite metoden* som er en metode, der kun placerer de højst rangerede varianter i matingpoolen, mens *probability metoden* kører efter et tilfældighedsprincip, når forældre placeres i puljen. Som tidligere beskrevet er en af de vigtigste faktorer for at udvikle en generation, der er bedre end den tidligere, at der er variation nok i populationen, og de mest fitte føre generationen videre – survival of the fittest princippet.

I denne løsningstilpassede GA er der som udgangspunkt brugt *probability metoden* for at sikre nok variation, men for ligeledes at sikre de bedste varianter har større mulighed for at blive valgt, er ”lykkehjul”-princippet som beskrevet i Afsnit 8.3 brugt.

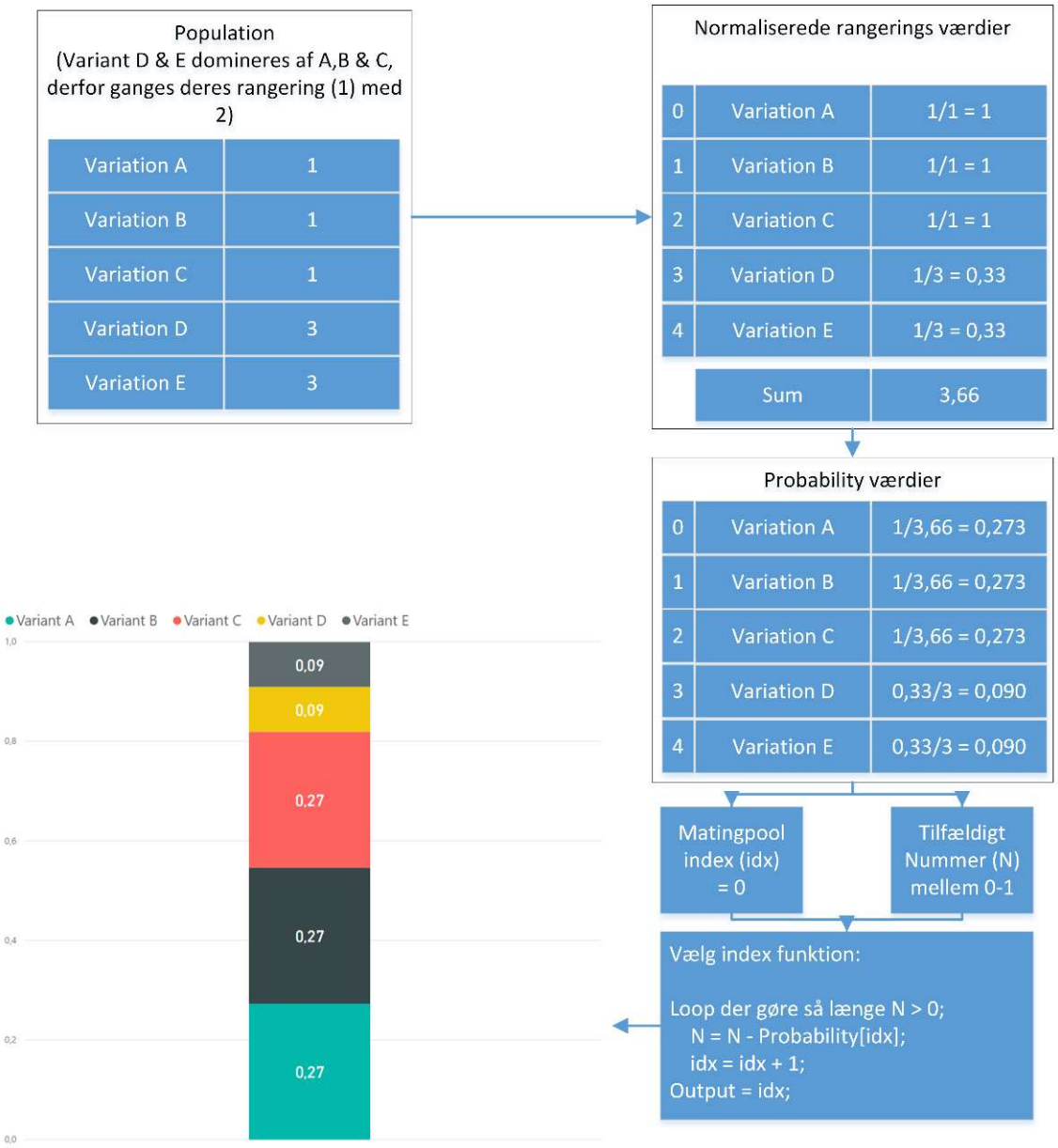
Igen er en custom python funktion oprettet til at genere en matingpool. Ligesom med Pareto-rangeringsfunktionen kræver matingpool-funktionen variationernes fitnessværdier som inputs. Funktionen generer derefter en række integers, der repræsenterer en indeks på den variant, de skal vælges som forældre. Som det ses på Figur 11-28, er det tilfældigt hvilken variant, der vælges, ligesom det ikke kun er varianter med den bedste rangering, der kan vælges. Dog er der indbyggede funktioner, der gør, at jo bedre rangeringen er jo større sandsynlighed for at blive valgt.



Figur 11-28 Workflow for generering af matingpool

Python-scriptet, der generer matingpoolen, består overordnet af to loops og en specielt oprettet funktion. Det første loop i scriptet er på mange måder det samme som Pareto-rangeringsfunktionen (Figur 11-27), men i stedet for tillægge 1 til rangeringen hver gang varianten domineres af en anden, ganges den nuværende rangeringsværdi i stedet med 2. På denne måde skabes der et større spænd mellem de bedst rangerede og de værste. Værdien lagres i optionRanking-listen en værdi for hver variant, der er i populationen.

Den næste loop i scriptet benytter funktionen pickOne til at udvælge en variant, der skal tilføjes til matingpoolen. Dette sker ved at vælge et tilfældigt indeks. Selvom varianten vælges tilfældigt, gør probability-metoden, at bedre rangerede varianter vælges oftere end dårligere rangerede. Probabilityscoren laves ved først at normalisere alle rangeringsværdier (se Figur 11-29 og Figur 11-30) ved at dividere 1 med rangeringsværdien. Efterfølgende sammenlægges alle de normaliserede rangeringsværdier, så hver normaliseringsværdi kan divideres med den samlede score. Dette tal giver variantens sandsynlighed for at blive valgt. Et tilfældigt nummer samt et indeksnummer generes. Disse to tal bruges til at finde det endelige indeksnummer, og dette sker ved hjælp at et loop, der kører så længe, det tilfældige nummer (N) er større end 0. For hver iteration af loopet fratrækkes probability-værdien af variationen på indekset = idx. Efterfølgende tillægges 1 til idx-variablen. Outputtet af funktionen er idx-værdien, som svarer til et indeks i populationen, og varianten på dette indeks overføres til matingpoolen.



Figur 11-29 Workflow diagram over matingpoolfunktionen

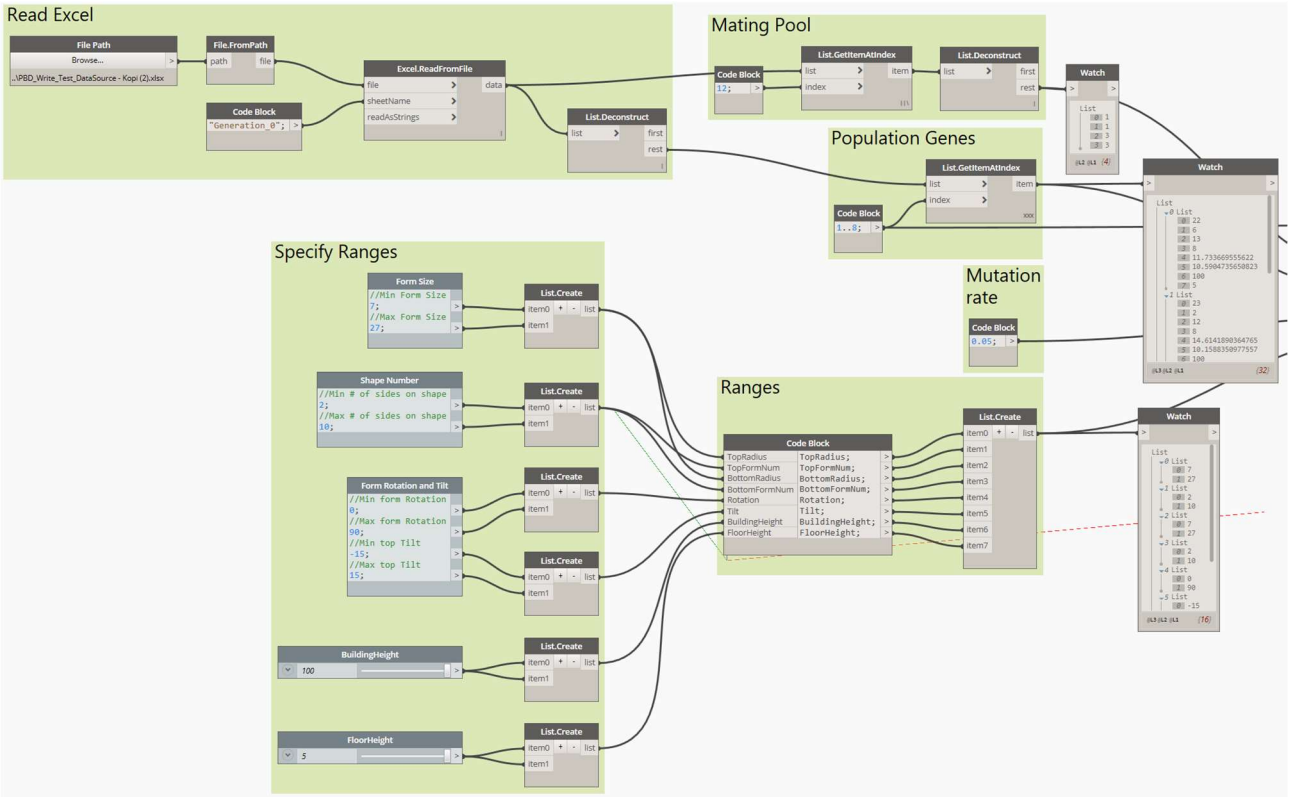
```python
import sys

pyt_path = r'C:\Program Files (x86)\IronPython 2.7\Lib'
sys.path.append(pyt_path)

import random

def pickOne(ranking):

    rankingScore = []
    for rank in ranking:
        rankingScore.append(1/float(rank))

    rankingScoreSum = sum(rankingScore)
    propbability = []
    for score in rankingScore:
        propbability.append(score/rankingScoreSum)

    pickIndex = 0
    randomN = random.uniform(0,1)

    while randomN > 0:
        randomN = randomN - propbability[pickIndex]
        pickIndex += 1
    pickIndex -= 1
    return pickIndex

fitness = IN[0]

optionRanking = [1] * len(fitness)

for i in range(0, len(fitness)):
    optionCounter = 0
    for j in range(0, len(fitness)):
        if fitness[i][0] > fitness[j][0] and fitness[i][1] > fitness[j][1]:
            optionRanking[optionCounter] += (optionRanking[optionCounter]*2)
        optionCounter += 1

output = []
for i in range(0, len(fitness)):
    output.append(pickOne(optionRanking))

OUT = output
```

Figur 11-30 Python-script for matingpoolfunktionen

11.4.3 CROSSOVER OG MUTATION

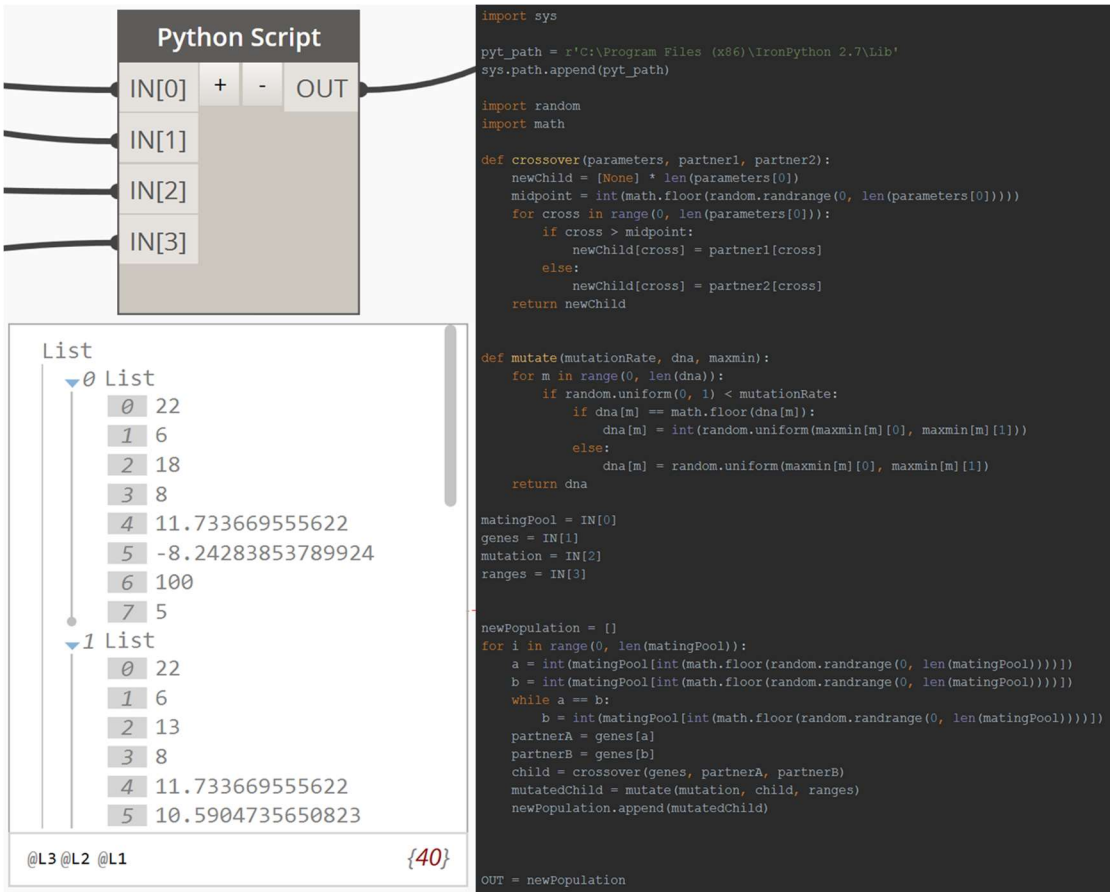
For at skabe en ny generation skal de udvalgte forældre fra den tidligere generation parres for at skabe nye børn, der kan danne populationen i den næste generation. Dannelsen af den nye population sker med afsæt i to principper, henholdsvis crossover og mutation. Crossover er funktionen, der finder to forældre fra matingpoolen, mikser generne fra forældrene og på den måde skaber et nyt barn. Mutation sikrer høj variation i populationen. Når et barn er skabt på baggrund af crossover, kan mutation ændre tilfældige gener og derved skabe en ny unik variant. I en GA er det vigtigt at opnå en ordentlig balance mellem udforsknings- og udnyttelsesevnen for optimeringsalgoritmen. Mutation hjælper med at dække et større design space, da nye mulige løsninger dukker op, som nødvendigvis ikke var mulige gennem den oprindelige population. Crossover står for udnyttelsesevnen, da denne funktion koncentrerer sig om at udvikle de løsninger, der allerede er lavet. Mutation angives i en GA som en sandsynlighedsprocent. Ofte skal denne procent være i den lave ende, og typisk ser man en mutaionsrate på mellem 1-10%, hvor 10 ofte kan blive for meget.

I løsningens GA sker crossover og mutation først i script to, fordi den indledende population først skal oprettes. Script to er oprettet som et stort loop, hvor crossover og mutation sker x antal gange. Selve crossover og mutationsfunktionen kræver en række inputs for at kunne genere den næste generation. Første step er at indlæse data fra den indledende population. Dataene sorteres i to lister; en med matingpool indeks og en med de enkelte kromosomers gener. Derudover tilføjes en mutaionsrate og der generes forskellige værdiintervaller i form af en minimums og maksimumsværdi.



Figur 11-31 Input workflow til crossover/mutationsfunktionen

Endnu en custom python-funktion er oprettet til at udføre crossover og mutation. Funktionen består af et loop af to specielt oprettede funktioner; en funktion til at udføre crossover og en til mutation. Loopet kører det samme antal gange, som der er kromosomer i populationen. For hver gang loopet køres, udvælges to tilfældige varianter fra matingpoolen. Varianterne gemmes som PartnerA og PartnerB (Figur 11-32). Når to forældre fra matingpoolen er udvalgt, kan selve crossoverfunktionen ske. Crossoverfunktionen kræver tre argumenter; *parameters* som er en liste med et kromosoms gener samt *partner1* og *partner2,* som er de to forældrekromosomer, som det nye skal skabes ud fra. Funktionen vælger tilfældigt et midtpunkt, som bestemmer, hvor mange gener, der skal komme fra partner1 og partner2. Hvis midtpunktet eksempelvis er tre, vil det nye kromosom bestå af de første fire gener (indeks starter med 0) fra partner1 og de sidste fra partner2. Når crossover er udført, sker mutationsprocessen gennem den anden funktion i scriptet. Mutationsfunktionen kræver ligeledes tre argumenter; en mutationsrate, det nye kromosom som var produktet af crossover funktionen og intervaller for alle genværdier. Funktionen består af et loop, der kører så mange gange, som der er gener i kromosomet. For hvert gen laves et tilfældigt nummer mellem 0 og 1, og dette nummer tjekkes i forhold til mutationsraten. Er det mindre, sker mutationen, hvis ikke returneres den originale værdi. Når mutationsfunktion er kørt, returneres det nye kromosom til en liste, som, når en ny generation er færdigudviklet, outputtes.



Figur 11-32 Crossover/Mutation custom Python-funktion

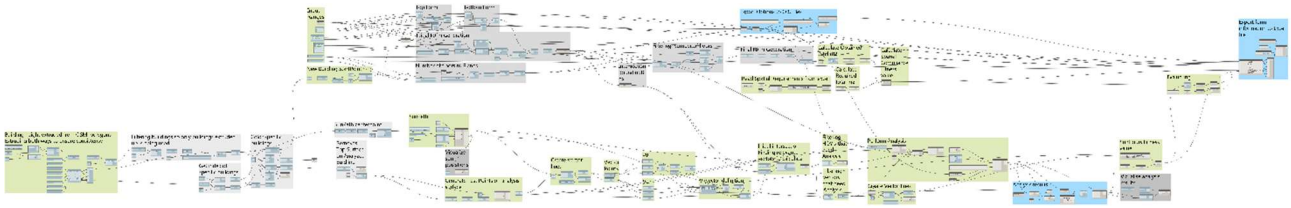
12 LØSNINGSPROCES

Det foregående afsnit beskrev i detaljer, hvordan løsningen teknisk er opbygget, de enkelte funktioner der inkluderer parametrisk design, generativt design og performance based design. I dette afsnit vil selve processen i forbindelse med den tekniske løsning beskrives. Udvikling og implementering af nyt software eller tekniske workflows vil ofte føre nye arbejdsgange og processer med sig, både på det tekniske plan men også med brugerens anvendelse. I dette projekt har fokus været på den tekniske del af processen, og det vil i dette afsnit derfor være den tekniske proces, der vil blive gennemgået.

12.1 TEKNISKE PROCES

Som tidligere beskrevet er løsningen opdelt i to forskellige scripts, hvor det første opretter den indledende population, mens det andet udvikler et givent antal populationer i et stort loop.

Script 1 (Figur 12-1), som er beskrevet i detaljer i afsnit 11, skaber den første population af mulige løsninger. Simpelt fortalt generer scriptet et brugerangivet antal forme på baggrund af nogle parameterintervaller ligeledes angivet af brugeren, som efterfølgende analyseres på baggrund af nogle objectives. Til sidst evalueres hver designvariation i forhold til deres Pareto-rangering, hvorefter der generes en matingpool. Alt data, der oprettes, eksporteres til et Excel ark. På denne måde kan man inspicere dataene efterfølgende, ligesom man kan følge udviklingen af generationerne. Ved at eksportere data fra hver generation kan udviklingen ikke blot følges, det er også muligt at lave en rangering på tværs af alle generationer, og dermed kan variationer, der ikke er udviklet i sidste generation også vurderes. Udover at eksportere data til Excel eksporteres hver variants geometri også gennem .obj-filformatet, så det er muligt efterfølgende at gense designvariationer. Dette gør det nemmere at vurdere designvariationer, så hvis flere variationer rangeres som 1, kan det være nødvendigt at kigge på parametre som æstetik, som ikke kan programmeres eller måles som en fitnessfunktion, men i stedet skal bruge menneskelig vurdering.



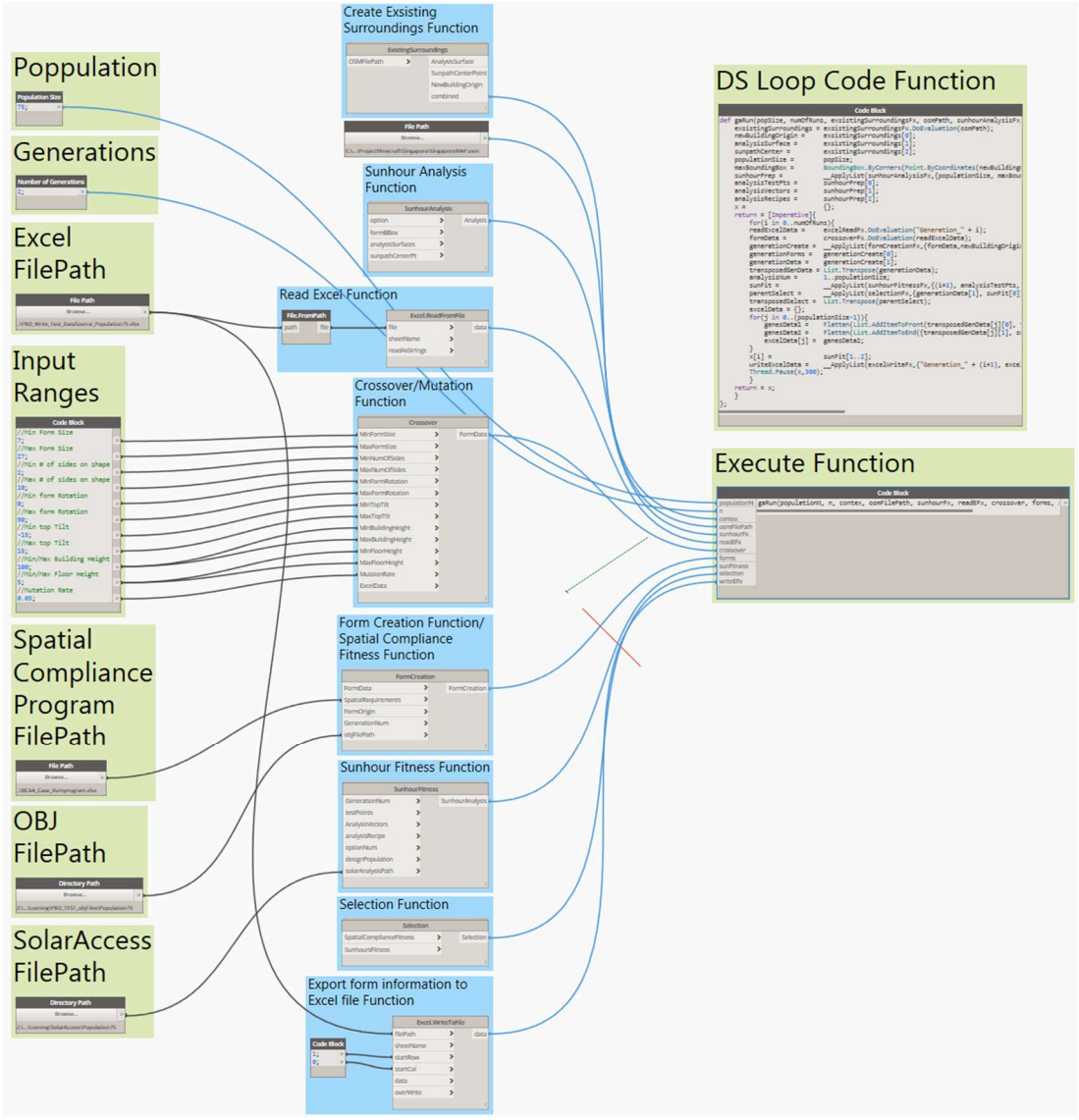
Figur 12-1 Oversigt over hele script 1 workflowet

Princippet bag en GA er hele tiden at udvikle nye generationer med designvariationer. Hver gang en ny generation skal udvikles, skal data fra den tidligere generation bruges. Det betyder, at der er to fremgangsmåder at udvikle generationerne på, enten køres et script manuelt for hver generation, hvor inputdataene ændres hver gang, der startes en ny generation, ellers køres et script i et loop, der automatisk bruger den rigtige data. Den sidste løsning vil altid være at foretrække, da det at udvikle optimale løsninger kan tage mange generationer. Gennem den manuelle proces ville man skulle vente, til hver generation er færdig for så at starte en ny generation. Dette er ineffektivt og kan spilde meget tid.

Løsningen er som tidligere beskrevet udarbejdet i det visuelle programmeringsmiljø Dynamo. Visuel programmering giver mange fordele især i forhold til tilgængelighed for ikke-programmører. Dog er der den store ulempe, at visuelle programmeringsworkflows ikke kan køres i et loop, hvor input data automatisk

opdateres. Der findes forsøg på iterativt at køre en Dynamo graph, hvor inputs ændrer sig ved hver iteration. Autodesks generative designværktøj Project Fractal (Fractal 2017) er et værktøj, der iterativt kan gøre en dynamo graph. Project Fractal er webbaseret, hvilket mindsker behovet for desktop computerkræfter. Ulempen ved at bruge Fractal er, at tredjeparts packages, såsom Ladybug Tools eller custom Python-funktioner, ikke er understøttet. Dette gør funktionaliteten af Fractal begrænset, da kun "out-of-the-box" Dynamo nodes kan benyttes. Et andet forsøg på iterativt at køre en Dynamo graph, er packagen Optimo (Asl et al. 2015), der benytter Zero-Touch nodes programmeret i C#. Optimo benytter som dette projekt en GA til at optimere et MOO problem. Ulempen ved Optimo er, at den grundet Zero-Touch er svær at tilpasse, medmindre man har indgående kendskab til Zero-Touch nodes og C# programmering.

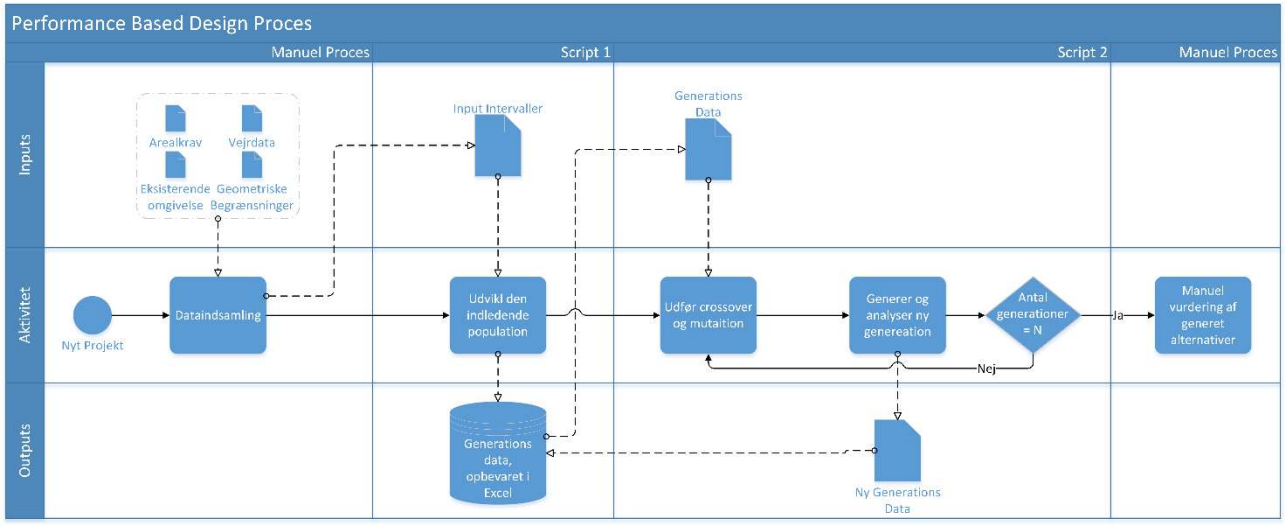
Denne løsning tager en ny tilgang til det at køre en Dynamo graph i et loop, en metode der umiddelbart ikke har været de store forsøg med. Metoden benytter Dynamo's eget programmeringssprog DesignScript (DS). DS er et tekstbaseret sprog, som man kender det fra Python og C#. Alle funktioner, der findes i Dynamo, er ligeledes tilgængelige i DS, og ligeledes er funktioner som *for* loops og *if* statements tilgængelige gennem DS.



Figur 12-2 DesignScript graph loop funktion

Ved at pakke dele af Dynamoscriptet ned i custom nodes er det muligt at bruge dem som funktioner. Sammenkobles disse funktioner med et DS loop, er det muligt at køre et script i et stort loop, hvor inputværdierne ændres ved hvert loop. Figur 12-2 viser, hvordan hele workflowet fra Figur 12-1 er pakket ned i forskellige custom nodes. Ved at undlade nogle inputs til de forskellige nodes behandles de som funktioner, der kan bruges præcis, som man kender det fra andre programmeringssprog. DS loopet i Dynamo oprettes som en stor funktion, som skrives i en codeblock. Fordi loopet er defineret som en funktion, kan det refereres i en ny codeblock. Denne codeblock minder om en normal node, som kræver en række inputs, som i dette tilfælde er en række funktioner, som behandles og giver et output.

Som beskrevet består selve workflowet af to forskellige script. Det første generer den indledende population, mens det andet kører en gentagende loop for at genere og udvikle nye generationer. Løsningens Performance Based Designproces består udover de to scripts også af to manuelle processer, hvor menneskelig faglighed er i fokus. De to manuelle processer er afhængige af specialister inden for områderne, der skal optimeres. I mange tilfælde vil en tværfaglig gruppe af specialister være nødvendig, da de opsatte optimeringsobjectives ikke nødvendigvis er fra det samme fagdomæne. De to manuelle processer består i at indsamle data og vurdering af de generede løsninger. Dataindsamling er processen, hvor alt det nødvendige data indsamles, og denne data skal bruges til at kvantificere de forskellige objectives, så det er muligt at bruge analyseoutputs som fitnessmål i den genetiske algoritme. Vurdering af de generede løsninger er en manuel proces, idet den genetiske algoritme ikke outputter én optimal løsning men i stedet en række løsninger, der ikke domineres af andre. Den endelige designbeslutning er altså op til designeren. Vægtningen af de enkelte objectives kan være afgørende for den endelige beslutning, ligesom ikke-kvantificerbare parametre som æstetik kan have stor indflydelse på beslutningen.



Figur 12-3 Workflow diagram over Performace Based Design processens tekniske løsning

13 TEST AF LØSNING

Projektets løsning er testet og evalueret gennem to metoder; en test af løsningen, hvor konsekvensen af forskellige ændringer i workflowet vurderes samt en præsentation af løsningen for et specialistpanel.

Første del af dette afsnit præsenterer den praktiske testdel, hvor fokus primært har været på at optimere den genetiske algoritme. Anden del af afsnittet tager udgangspunkt i specialistpanelets kommentarer til projektets løsning.

13.1 LØSNINGSTEST

I forbindelse med et workflow som det præsenteret i Afsnit 11 er der mange parametre, der kan justeres i forhold til optimering og bedre performance. Løsningen består af et workflow, hvori mange elementer er koblet sammen. Hvert af disse elementer kunne testes i forhold til resultater og performance, men de fleste elementer af løsningen skal dog ses som værende projektspecifikke, og derfor fokuseres der i dette afsnit på test af den genetiske algoritme.

Inden for den genetiske algoritme (GA) kan der ligeledes kigges på flere dele, der kan testes og optimeres. Selve koden, der udgør en GA, kan på flere måder testes og optimeres, hvor optimeringen af denne har forgået i forbindelse med oprettelsen af den. Der findes mange måder at udføre de forskellige elementer i en GA på. Flere metoder er afprøvet, hvorefter der på baggrund af forskelligt litteratur og egne tests er valgt en metode. Begrundelsen for, hvordan GA'en er kodet, vil ikke yderligere blive gennemgået eller testet. I stedet fokuseres der på det større billede. Med det større billede menes der, at der kigges på, hvordan GA'en, kan optimeres i forhold til det samlede workflow.

Som tidligere beskrevet er variation en nøglefaktor i en GA. Uden variation kan der ikke ske evolution, uden evolution forbliver løsningerne de samme. Det vigtigste element i at opnå høj variation er størrelsen på populationen. Jo flere elementer i en population jo bedre er chancen for bred variation. Dog handler opbygningen af et workflow med en GA-tilgang om balance mellem størrelsen af populationen og performance af workflowet. Jo større population, jo mere variation men også dårligere performance, når workflowet køres. Omvendt giver en mindre population bedre performance men dårligere variation. Litteraturen er mangelfuld, når det gælder undersøgelser inden for den optimale populationsstørrelse. Nogle forsøg har vist, at antal parametre x 10, giver et godt udgangspunkt for en population med god variation, mens andre argumenter for, at denne regel ikke er optimal.

Mutation er den anden måde at skabe variation i en population, og igen er det vigtigt at finde en balance mellem for meget og for lidt mutation. Det anbefales ofte ikke at overstige en mutationsrate på 10%, og en rate på 1% betragtes som værende minimum. Intervallet mellem 1-10 % vurderes derfor som værende et godt udgangspunkt.

Denne løsningstest kigger på, hvordan populationsstørrelsen har indflydelse på evolutionen af løsninger. Inden testene er påbegyndt, er der foretaget et udgangspunkt, som gør sig gældende ved hver test. Som udgangspunkt sættes mutationsraten til 5% i alle tests, og derudover genereres der 20 generationer ved hver test. Alle tests udføres med udgangspunkt i det udleverede testprojekt fra casen. Objectives, parametre og deres intervaller er derfor det samme som det beskrevet i Afsnit 11.

For at teste løsningen er der lavet fem tests, der alle bruger den samme geometri og analysealgoritme samt samme GA-opbygning. Det eneste, der varierer i de forskellige tests, er populationsantallet og enkelte parameterintervaller.

Formålet med denne test er at undersøge konsekvensen af populationsstørrelsen. De fem tests udgangspunkt er illustreret i Tabel 13-1.

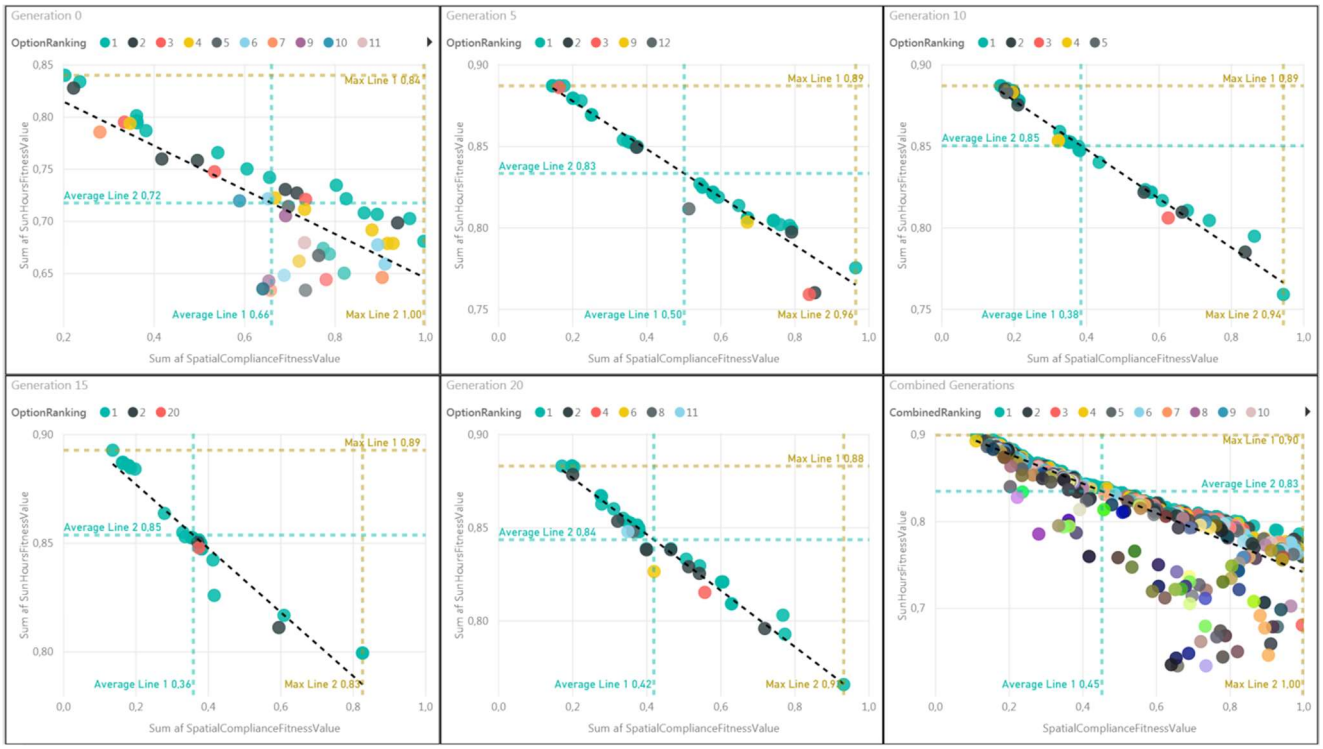
I test 4 og 5 er der forsøgt at ændre intervallet i størrelsesparametrene. I test 1-3 er intervallet i størrelsesparameteret en integer mellem 7-27, hvilket svarer til, at størrelsen kun kan ændres med 1 meter ad gangen. Dette kan forringe evnen til at optimere designet, da et spring på 1 meter hurtigt bliver for meget. Derfor er der i test 4 og 5 ændret på dette, så intervallet i stedet er et floating point mellem 7-27, hvilket giver bedre mulighed for mindre justeringer.

Test #	Populations størrelse	Generationer	Antal variationer	Bemærkninger
Test 1	50	20	1000	
Test 2	75	20	1500	
Test 3	100	20	2000	
Test 4	100	20	2000	Parameterintervaller der styrer formens størrelse, ændret fra integer til floating point, for bedre finjustering.
Test 5	200	20	4000	

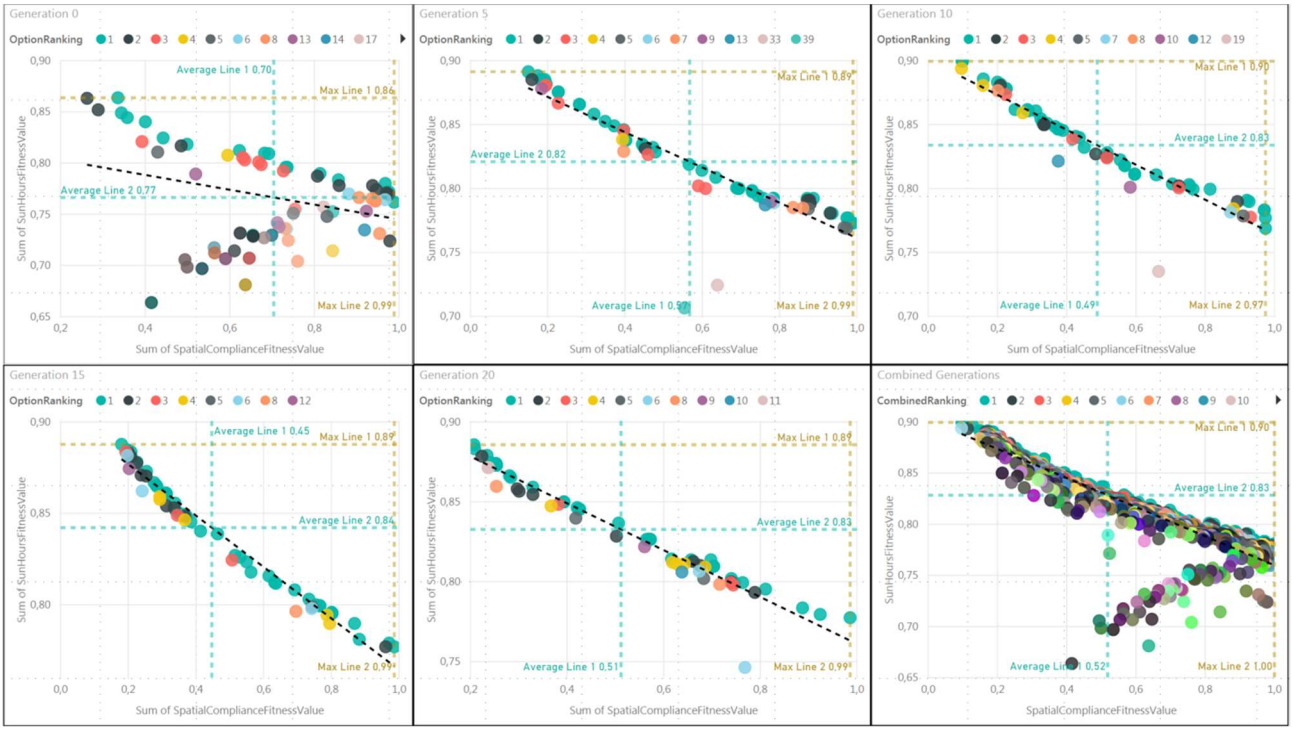
Tabel 13-1 Populations størrelse test

Figur 13-1 til Figur 13-5 illustrerer udviklingen af generationerne gennem de forskellige populationsstørrelser. Fælles for alle populationerne er den første generations spredning, som sker grundet tilfældighedsprincippet, og alle varianter generes altså tilfældigt. Allerede efter generation 5 lægger variationerne sig på en mere lineær kurve, varianterne bliver mere koncentrerede og udvikler sig i den rigtige retning, og derfor opstår den mere lineære tendens. Det er ligeledes interessant at konstatere, at variationen i outputtet af de forskellige populationer er meget sammenlignelige. Især ved at sammenligne "Combined Generations"-feltet, som sammenlægger alle generationer og foretager en ny rangering, kan det ses, at den øgede population i dette tilfælde har minimal effekt.

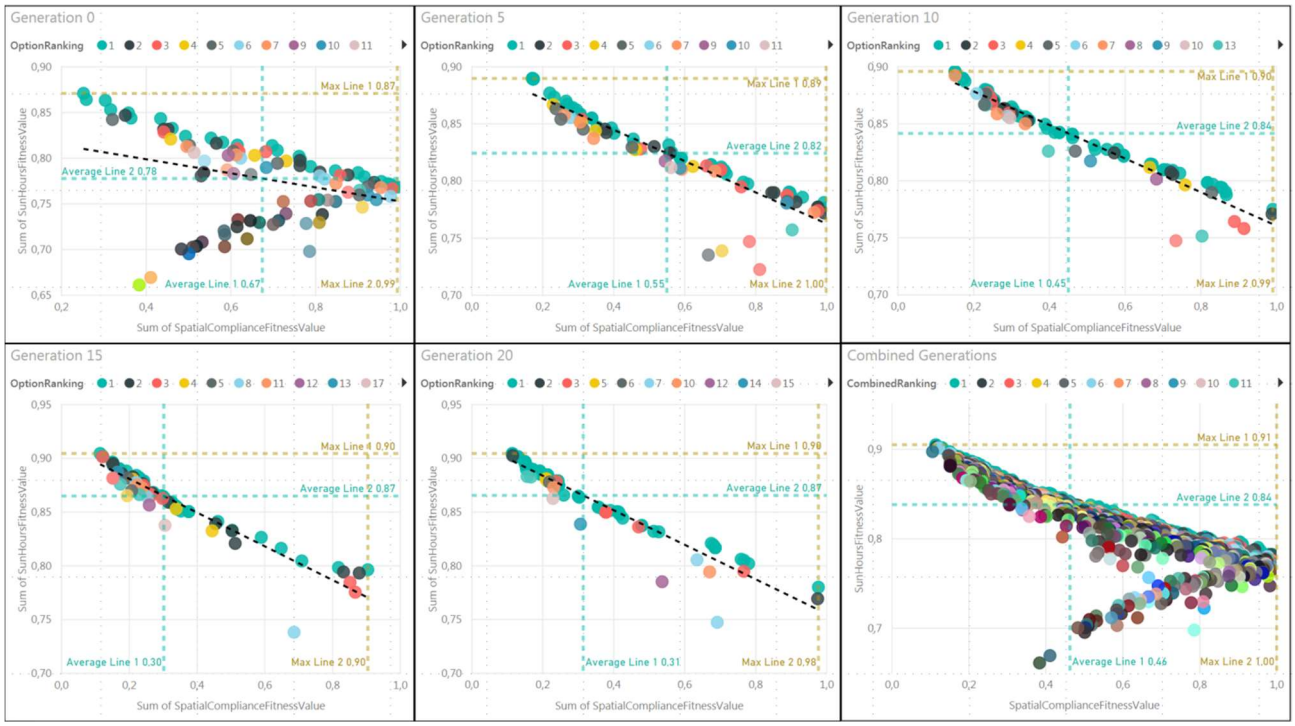
Disse forsøg lægger sig i høj grad op ad populationsteorien, der siger antal parametre x 10, som i dette tilfælde giver en optimal population på 50. Den ekstra gevinst, der ligger i større populationer end 50-75, har vist sig at være minimal. Dette afspejler sig i antallet af parametre, der har haft indflydelse på algoritmen. Flere parametre og objectives ville give et større behov for flere varianter i hver population.



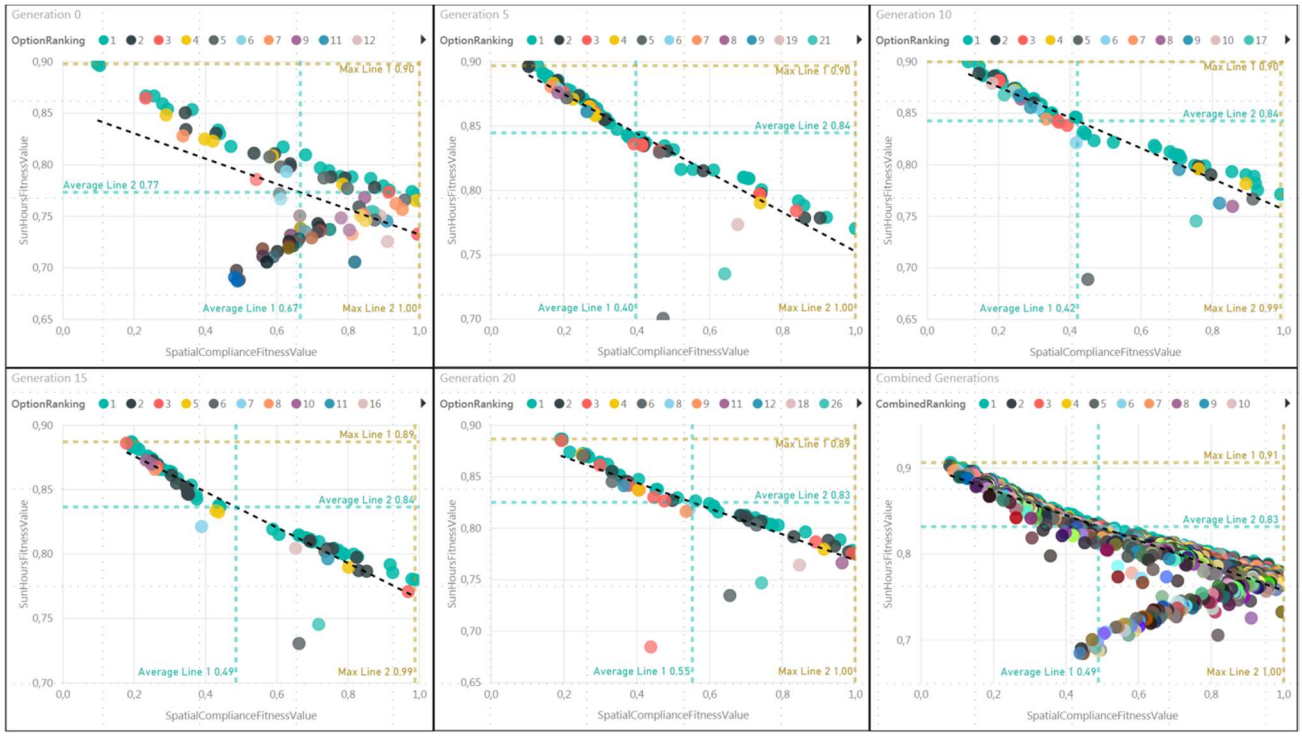
Figur 13-1 Generationsudvikling, population 50



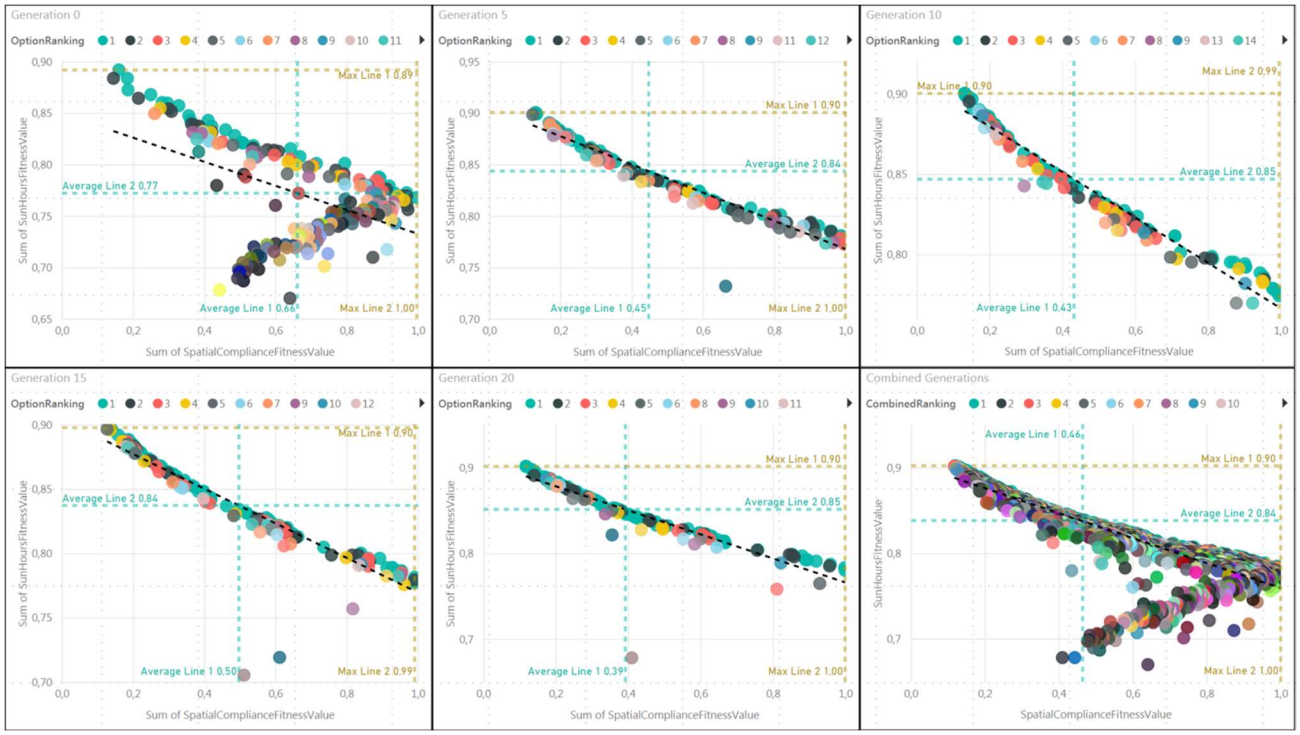
Figur 13-2 Generationsudvikling, population 75



Figur 13-3 Generationsudvikling, population 100

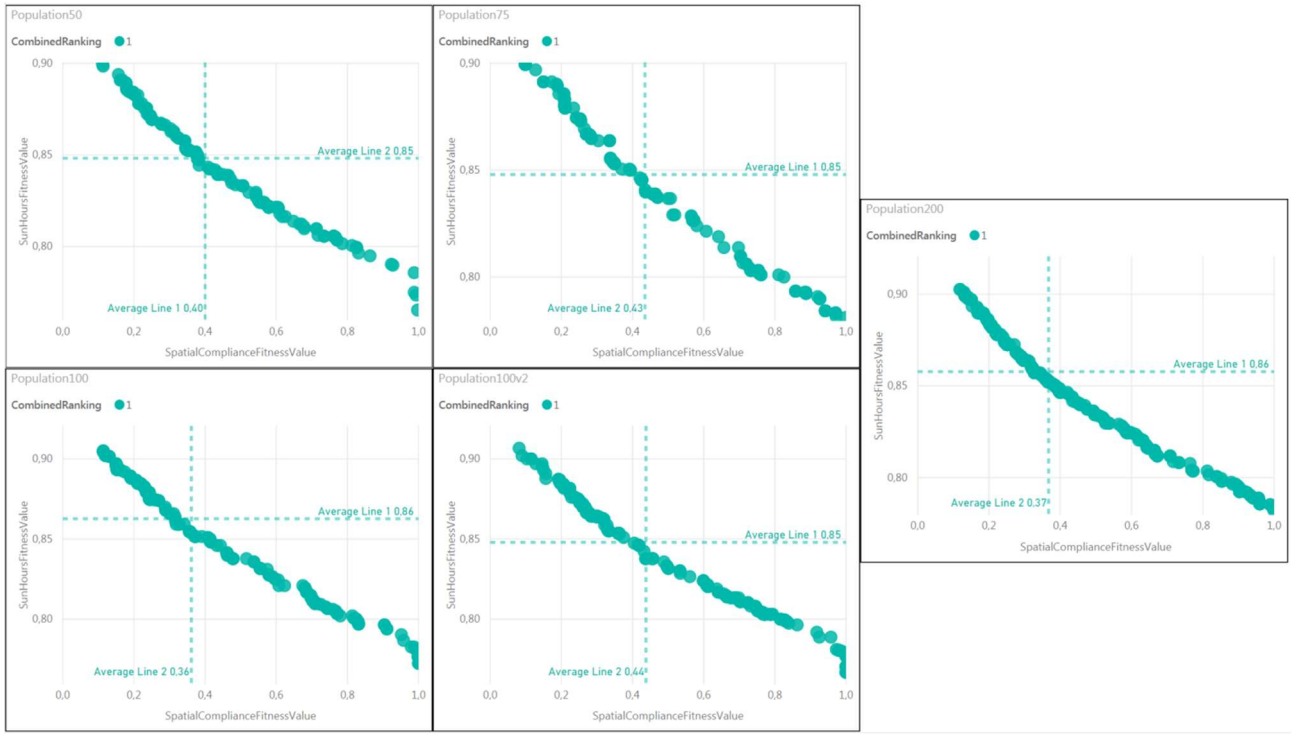


Figur 13-4 Generationsudvikling, population 100 v2



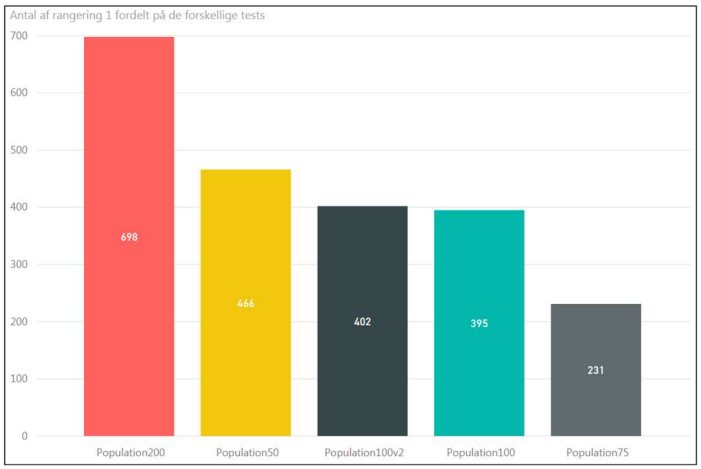
Figur 13-5 Generationsudvikling, population 200

Ved at filtrere graferne så kun variationer med rangeringen 1 er synlige (Figur 13-6), er det ekstra tydeligt, at større populationsantal ikke er bedre i dette tilfælde. Alle kurverne er meget sammenlignelige, både i deres max og min, men også i averagetendensen. I population 100 v2, hvor parameterintervallerne er justeret, ses der en lille forbedring i resultatet, hvilket tyder på, skridtene i intervallerne er vigtige. En populationsstørrelse med justerede intervaller vurderes derfor at kunne give det optimale resultat.



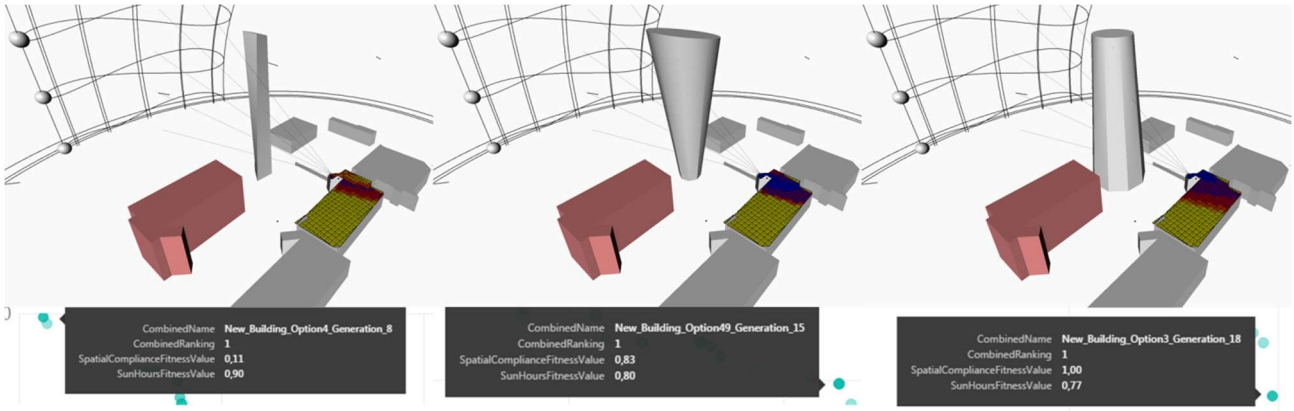
Figur 13-6 Samlede generationer, variationer med rangering på 1

Antallet af varianter med rangeringen 1 er naturligt højere i de større populationer, dog inkluderer nedenstående graf (Figur 13-7) også duplikerede varianter. Der kan altså være varianter, der går igen flere gange. En bedre indikator på, at den mindre population er tilstrækkelig, er derfor Figur 13-6, som viser populationens Pareto-front, hvor der også kan ses intervallet, som varianterne spænder over.



Figur 13-7 Antal designvarianter med rangeringen 1, fordelt på populationsstørrelser

At vælge den rigtige variation er et spørgsmål om vægtningen af de forskellige målte objectives, ligesom ikke-målbare elementer kan komme i spil, hvor æstetik er svært at måle på og kræver menneskelig interaktion. Alt efter hvordan objectives vægtes, kan der kigges i de to ekstreme ender max soltimer eller max arealoverholdelse. Alternativt kan der kigges på kompromiser mellem de to.



Figur 13-8 Visualisering af variationer, fra en yderlighed til en anden

13.2 LØSNINGSEVALUERING

Anden del af løsningstesten består af en evaluering fra en specialistgruppe. Evalueringen er foretaget via en præsentation af løsningen og visionen bag for fem VDC specialister fra MT Højgaard. Præsentationen er foregået på et teknisk niveau, der samtidig har præsenteret perspektiverne for løsningen, og hvordan den kan implementeres på forskellige projekter. Efter præsentationen er der indgået en dialog om fremtidsmuligheder, forslag til forbedringer, og hvordan løsningsworkflowet kan ændre designprocessen.

Med fokus på den tekniske opbygning af workflowet blev der bl.a. spurgt ind til om formgenereringen kunne gøres mere generisk på en måde, så kun volumenbegrænsninger skulle angives. De indledende tanker bag projektet var netop at kunne generere formen på denne baggrund, men flere forsøg viste dog, at dette ikke nødvendigvis var en holdbar metode, hverken teknisk eller teoretisk. Teknisk set er det meget svært at udføre en algoritme, der kan håndtere en formgenerering på denne måde. Udover det tekniske blev det også diskuteret, at det at kunne specificere et geometrisk design space er hensigtsmæssigt i forhold til at tage højde for konteksten, som bygningen placeres i. Ud over formgenereringselementet i algoritmen blev objectivesdelen diskuteret. Nogle af bekymringerne var det arbejde, der ligger bag disse, som på hvert projekt skal startes på ny. Hvert projekt er forskelligt på mange parametre, men alligevel vil der være nogle overordnede parametre, som er styrende for et design. En af udviklingsmulighederne, der blev diskuteret, var at identificere en række af disse objectives, som kunne pakkes ned i en generisk pakke, som kan bruges fra projekt til projekt. At have et bibliotek, hvor forskellige grundelementer kan plukkes fra, vil gøre workflowet mere automatiseret, så der i sidste ende kan spares tid eller bruges tid på andet.

Udover den tekniske del af løsningen blev der ligeledes diskuteret, hvordan løsningens workflow vil have indflydelse på den nuværende designproces. I dag er designprocessen typisk en proces, der analyseres, genereres tegninger og modelleres samtidigt, hvor de forskellige dele af processen sker manuelt og gentages ved hver ændring. Med et workflow som det præsenteret i dette projekt åbnes der op for

muligheden for at kunne automatisere og adskille disse processer. En proces, hvor designanalyse og modellering er sammenlagt men adskilt fra tegningsprocessen, giver et mere fleksibelt workflow, der kan minimere gentagne ændringer. Samtidig åbner det op for automatiseringsperspektiver, hvor form og analyseprocessen allerede er automatiseret, kunne man forstille sig at tegningsproduktionen i højere grad også kunne automatiseres.

14 KONKLUSION

Nærværende rapport beskriver én tilgang til en løsning af den opsatte problemformulering, beskrevet i starten af rapporten. Projektet har været fokuseret indenfor et løsningsdomæne, hvor flere forskellige teknologier og tilgange er sammensat til et stort workflow. For at besvare hovedproblemstillingen, er der opsat tre underspørgsmål, der er undersøgt og besvaret.

Hvordan kan man udforske flere alternativer i et design space, samt optimere metode til at søge igennem et design space?

Et gennemgående tema i dette projekt har været design space udforskning. Flere alternative og analyserede designalternativer, ses af mange forskere og organisationer indenfor branchen, som værende en nøglefaktor for bedre projekter. Flere undersøgelser viser at der ligger et kæmpe uudnyttet potentiale, i bedre udnyttelse af et projekts design space. Samtidig viser undersøgelser også at byggebranchen er alt for dårlige til håndtere flere alternativer. Undersøgelsen lavet af det amerikanske forsknings- og rådgivningsfirma Gartner, beskrevet i afsnit 1.1, viser at der i snit kun arbejdes med 2,7 velanalyserede alternativer. For at kunne øge antallet af analyserede alternativer i et projekt, er der brug for metoder der kan håndtere den store beregningsmæssige belastning, det kræver at analysere flere tusinde alternativer. Parametrisk design har i mange år været brugt som metode, til at kunne foretage iterativt design, hvor et design kan ændres på få sekunder ved blot at ændre på enkelte parametre. Selvom parametrisk design på mange måder har udvidet og flyttet grænserne for menneske-maskine interaktion i designprocessen, er design space udforskning stadig begrænset til menneskets evner og ressourcer. Ændringerne sker stadig som en manuel proces, hvor parametre ændres et efter et, hver gang et nyt design skal generes. Udforsknings processen er stadig afhængig i designerens intuition, og processen er derfor stadig meget lignende en traditionel design udforsknings proces. I dette projekt er generative design konceptet, undersøgt som metode til bedre at udforske et design space, ved at flytte den manuelle designer proces til en automatiseret computer proces. Computeren kan behandle informationer langt hurtigere end mennesker, hvilket gør det muligt at foretage en dybere udforskning af et komplekst design space. Generative design benytter som grundlag en parametrisk model, dog kræves det at den parametriske algoritme udvides på to fronter. For det første er det nødvendigt at inkludere målbare objectivefunktioner, så hvert design kan evalueres. Disse objectives er vigtige da computeren ikke har nogen eksisterende intuition om design, designeren er derfor nødt til eksplicit at beskrive hvordan design evalueres overfor computeren. Derudover skal den parametriske algoritme inkludere en søge- eller optimeringsalgoritme, der bliver styrende for hvordan input parametrene intelligent fintunes, så nye alternativer udvikles. En lovende metode til at søge igennem et design space, er gennem en genetisk algoritme, som bruger principperne om evolution til at udvikle designgenerationer, der hele tiden optimeres i forhold til objectives. Ved at kombinere en parametrisk model og generative design, med en genetisk algoritme, opnås en effektiv metode til at gennemsøge et design space og derved muliggøres en evaluering af langt flere alternativer end ved traditionelle metoder.

Hvordan kan designproblemer optimeres ud fra flere parametre på samme tid?

Designproblemer består typisk af mange parametre, der hver især har indflydelse på hinanden. Det betyder at et design ikke blot kan analysere med udgangspunkt i en

specifik analyse. De forskellige analyser der laves på et design, kan i mange tilfælde have en negativ indflydelse på hinanden, en optimering på baggrund af en analyse kan altså have negativ indflydelse på et andet analyse resultat. Performance Based Design (PBD) og Multi Objective Optimization (MOO), har i dette projekt været drivende for løsningen. PBD skaber et design på baggrund af performance simuleringer, analyserne er altså drivende for designet og sker i forbindelse med formgenereringen, fremfor at være et dokumentations element der sker efter formen er fastlagt. For at PBD bliver rigtig værdifuldt, er det nødvendigt at integrer flere analyser på samme tid. At optimere på baggrund af flere analysere på engang, kaldes MOO og er en vigtig faktor i PBD. MOO optimere et design ud fra en samlet helhed, af de forskellige performance analyser der foretages. Dette gør at de bedste design kompromisser findes, der hvor alle objectives er optimeret så meget som muligt. Ved at integrer disse analyser med selve formgenereringen, kan effektivt workflow opnås, i modsætning til traditionelle workflow hvor analysedelen og formgenereringen forgår i forskellige miljøer. Dette leder videre til det sidste underspørgsmål.

Hvordan kan åbne standarder bruges til at skabe et workflow, der kan måle sig med tilsvarende lukkede software?

Traditionelt set er analyse- og designsoftware opdelt i forskellige programmer, som primært er forbundet via envejskommunikation, hvor designmodellen konventernes til en analysemodel som der foretages analyser på. Hvis designmodellen skal redigeres i forhold til analyserne, skal det ske i designsoftwaret, hvorefter der igen skal genereres en ny analysemodel. Dette workflow er ineffektivt og leder til suboptimeringer på baggrund af enkelte analyser. I dette projekts løsning, er der skabt et workflow der integrer formgenerering, analyser og optimering i det samme miljø. Workflowet er skabt som en *Integrated Dynamic Model,* hvor analyser værktøjer kobles sammen med formgenerering, gennem Dynamo som middleware. Kobling sker gennem forskellige komponenter, der gør det muligt at benytte analysemotorer direkte i det visuelle programmerings miljø. Ved at benytte komponenter til at sammensætte et workflow, gives der mere frihed til designeren, og der skabes et bedre beslutningsgrundlag da analyserne foretages samtidig med der modelleres. Gennem disse komponenter skabes en værktøjskasse af software, der gør det muligt at bruge præcis det rigtige værktøj til hver opgave, og selv sammensætte de forskellige værktøjer, på den måde er det ikke de enkelte software der definere hvordan designprocessen forgår, men designeren selv.

Hvordan kan performance based design understøtte en mere datadriven beslutningsproces i den tidlige designfase?

For at besvare projektets problemformulering, har det vist sig at tre kerne elementer kan skabe en bedre beslutningsgrundlag i den tidlige designfase. For det første er det afgørende at lave en tilstrækkelige udforskning af et design space, dette sker ved at bruge generative design, sammen med en søge/optimeringsalgoritme. For det andet skal der foretages flere analyser på én gang, design udforskningen skal ses som MOO problem, hvor der optimeres på flere objectives. På den måde skabes en række alternativer, der på hver sin måde er optimale i forhold til de opstillede objectives. Til sidst er der behov for et miljø hvor åbne standarder kan integreres i den samme platform, og softwareværktøjskasser kan benyttes hvor de rigtige komponenter kan udvælges, så et samlet workflow kan oprettes. Projektets løsning har vist hvordan disse tre elementer, kan skabe et workflow hvor et design skabes, på baggrund af dets performance. Det er ligeledes vist hvordan dette gøres på en automatisk måde, hvilket kan skabe en hurtigere og bedre design proces.

15 PERSPEKTIVERING

Følgende afsnit vil perspektivere på projektets løsning, i en større sammenhæng. Der perspektiveres på elementer der ikke er behandlet i rapporten, som den ændrede arbejdsproces et workflow som dette giver, samt de fremtidige udviklingsmuligheder for løsningen. En af de helt store styrker i den udviklede løsning er skalerbarheden, alle elementer kan nemt tilpasse nye typer projekter, eller andre krav til designet. I dette projekt er der vist hvordan Generative Design, Multi Objective Optimizaton (MOO) og en Genetisk Algoritme kan udvikle et bygningsdesign, man kunne ligeledes forstille sig de samme principper blive brugt på mindre opgaver. De fleste valg der træffes i et byggeprojekt har indflydelse på andre dele, derfor er MOO særligt relevant i denne sammenhæng, valg af enkelte bygningsdele kunne være ideelt at bruge denne tilgang på.

Ved et workflow som det præsenteret i rapportens løsning, ændres ikke kun de teknologiske aspekter, men i høj grad også selve arbejdsprocessen omkring designet. Typisk ses byggeprojekter opdelt i siloer, hvor de enkelte aktører levere ydelser til projektet og hinanden, for at udnytte potentialet i et Performance Based Design projekt, skal være mere samarbejde og tidlig inddragelse. For at sikre at de rigtige objectives er med i design algoritmen, er det nødvendigt at have den rigtige specialist viden, fra alle relevante aktører. Workflowet ligger altså op til at samarbejdsformer som Integrated Project Delivery benyttes, hvor en af hovedelementerne er tidlig involvering. Ved at inddrage projektets aktører fra starten, kan de rigtige objectives bestemmes og udvikles, hvilket i sidste ende vil give et design der er udviklet på et bedre analyse grundlag. Udover et tætter samarbejde mellem de enkelte faggrupper, er der også behov for en teknologi specialist, der kan formulere de forskellige objectives igennem programmering. Selve det at køre løsningen kræver ligeledes en specialist, da fleksibilitet er en stor faktor, er brugervenlighed nedprioriteret. Det betyder at der er behov for en specialist med kendskab til algoritmisk design, til at sammensætte de forskellige dele i et samlet workflow.

Der ligger mange perspektiver i udviklingen af løsningen, mange af dem vil ligge som projektspecifikke udviklinger, men potentialet i at oprette forskellige standard objektives og form biblioteker er ligeledes stort. Jo mere erfaring der opnås med at bruge løsningen på projekter, jo mere viden opnås der omkring hvilke objectives der er vigtige i den tidlige design fase. Når denne vide er stor nok, kan der oprettes et bibliotek med standard objectives, som kan bruges direkte eller ændres til der specifikke projekt. Det samme gælder på geometrisiden, hvis det kortlægges hvilke forme der typisk bliver brugt, kan disse oprettes med de rette parametre så de ligeledes kan gemmes i et standard bibliotek. Ved at oprette disse biblioteker, vil løsningen få en højere grad plug-and-play, hvilket også vil gøre det mere brugervenligt. Dog vil der som udgangspunkt altid være et behov for specialist viden til at sammensætte det samlede workflow.

IV. AFSLUTTENDE SERVICE AFSNIT

16 LITTERATURLISTE

Aish, R., 2011. Designing at t + n. *Architectural Design*, 81(6), pp.20–27. Available at: http://doi.wiley.com/10.1002/ad.1315 [Accessed October 9, 2017].

Akin, Ö., 2001. Variants in Design Cognition. In C. Eastman, M. Mccracken, & W. Newstetter, eds. *Design Knowing and Learning: Cognition in Design Education*. pp. 105–124.

Akos, G. & Parsons, R., 2014. Foundations. The Grasshopper Primer Third Edition. , p.142. Available at: http://modelab.is/grasshopper-primer/.

Aksamija, A. & Zaki, M., 2010. Building Performance Predictions. *Perkins+Will Research Journal*, 2, pp.7–32.

Anton, I. & T□nase, D., 2016. Informed Geometries. Parametric Modelling and Energy Analysis in Early Stages of Design. *Energy Procedia*, 85(November 2015), pp.9–16.

Asl, M.R. et al., 2015. Optimo□: A BIM-based Multi-Objective Optimization Tool Utilizing Visual Programming for High Performance Building Design. *eCAADe: Conference of Education and Research in Computer Aided Architectural Design in Europe*, 1, pp.673–682.

Attia, S. et al., 2009. Architect Friendly: a Comparison of Ten Different Building Performance Simulation Tools. *Eleventh International IBPSA Conference*, pp.1–8. Available at: papers2://publication/uuid/C5D874D9-B854-4E97-980B-A555604BE791.

Attia, S. et al., 2012. Simulation-based decision support tool for early stages of zero-energy building design. *Energy and Buildings*, 49, pp.2–15.

Augenbroe, G., 2002. Trends in building simulation. *Building and Environment*, 37(8–9), pp.891–902.

Bogenstätter, U., 2000. Prediction and optimization of life-cycle costs in early design.

Burry, M., 2011. *Scripting cultures□: architectural design and programming*, Wiley.

Ceccato, C., 2012. Computing and Constructing Continuous Differentiation. In *Material Computation: Higher Integration in Morphogenetic Design*. John Wiley & Sons, pp. 96–103. Available at: http://doi.wiley.com/10.1002/ad.1385 [Accessed October 17, 2017].

Cross, N., 2001. Design Cognition: Results From Protocol And Other Empirical Studies Of Design Activity. In C. Eastman, M. Mccracken, & W. Newstetter, eds. *Design knowing and learning: cognition in design education*. pp. 79–103. Available at: http://www.elsevier.com/wps/find/bookdescription.cws.

Cross, N., 2006. *Designerly Ways of Knowing*, Springer-Verlag London.

Davis, D., 2013. Modelled on Software Engineering□: Flexible Parametric Models in the Practice of Architecture. *Ph.D Thesis*, (February), p.243. Available at: http://www.danieldavis.com/papers/danieldavis_thesis.pdf.

Dino, I.G., 2012. Creative design exploration by parametric generative systems in architecture. *Metu Journal of the Faculty of Architecture*, 29(1), pp.207–224.

Fractal, 2017. Autodesk. Available at: https://home.fractal.live/ [Accessed December 16, 2017].

Gerber, D.J. & Lin, S.-H.E., 2014. Designing in complexity: Simulation, integration, and multidisciplinary design optimization for architecture. *Simulation*, 90(8), pp.936–959. Available at: http://journals.sagepub.com/doi/10.1177/0037549713482027.

Gerber, D.J. & Lin, S.H.E., 2014. Designing-in performance: A framework for evolutionary energy performance feedback in early stage design. *Automation in Construction*, 38, pp.59–73. Available at: http://dx.doi.org/10.1016/j.autcon.2013.10.007.

Gero, J., 1994. *Advances in Formal Design Methods for CAD*,

Kolarevic, B., 2005. Computing the Performative in Architecture. In B. Kolarevic & A. Malkaw, eds. *Performative Architecture: Beyond Instrumentality, Routledge*. pp. 195–202.

Ladybug Tools, 2017. Ladybug Tools | Home Page. Available at: http://www.ladybug.tools/index.html

[Accessed November 23, 2017].

Leach, N., 2009. Digital Morphogenesis. *Architectural Design*, 79(1), pp.32–37. Available at: http://doi.wiley.com/10.1002/ad.806 [Accessed October 13, 2017].

Lynn, G., 1999. *Animate form*,

Mackey, C. & Roudsari, M.S., 2018. The Tool (s) Versus The Toolkit. , 2.

Moradi, M., 2014. Making-Intelligent Place of Buildings in Parametric (Algorithmic) Architecture. , 4(2), pp.103–109.

MT Højgaard, 2017. MT Højgaard. Available at: http://mth.dk/ [Accessed December 20, 2017].

Negendahl, K., 2015. Building performance simulation in the early design stage: An introduction to integrated dynamic models. *Automation in Construction*, 54, pp.39–53. Available at: http://dx.doi.org/10.1016/j.autcon.2015.03.002.

Obitko, M., 1998. Encoding - Introduction to Genetic Algorithms - Tutorial with Interactive Java Applets. Available at: http://www.obitko.com/tutorials/genetic-algorithms/encoding.php [Accessed October 20, 2017].

Oxman, R., 2008. Performance-Based Design: Current Practices and Research Issues. *International Journal of Architectural Computing*, 6(1), pp.1–17. Available at: http://journals.sagepub.com/doi/10.1260/147807708784640090.

Philip, E. & Philip, C.F., 2014. *Point-of-View on digital enterprise and BIM / VDC*,

Radsite, 2017. Radiance — Radsite. Available at: https://www.radiance-online.org/ [Accessed December 4, 2017].

Rothenborg, M., 2017. Boligmangel i København kan løses - Rambøll i Danmark. Available at: http://www.ramboll.dk/medier/rdk/boligmangel-i-kobenhavn-kan-loses [Accessed September 20, 2017].

Roudsari, M.S., 2016. Seeing the Process: Ladybug + Honeybee, Dynamic Building Simulation Solutions for Integrated Iterative Design. In D. Willis et al., eds. *Energy accounts□: architectural representations of energy, climate, and the future*. Routledge, pp. 112–116.

Shiffman, D., 2012. *The Nature of Code*,

Smith, R., 2007. Technical Notes from experiences and studies in using Parametric and BIM architectural software . Virtual Build Technologies. *Notes*.

Teknik & Miljø, 2006. *HØJHUSPOLITIK for Århus Kommune*,

Terzidis, K., 2011. Algorithmic Form. In A. Menges & S. Ahlquist, eds. *Computational Design Thinking*. Wiley.

Veldhuizen, D.A. Van et al., 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems*, Boston, MA: Springer US. Available at: http://link.springer.com/10.1007/978-0-387-36797-2 [Accessed October 24, 2017].

Wang, W., Zmeureanu, R. & Rivard, H., 2005. Applying multi-objective genetic algorithms in green building design optimization. *Building and Environment*, 40(11), pp.1512–1525.

Weisstein, E.W., 2003. *CRC concise encyclopedia of mathematics*, Chapman & Hall/CRC.

Østergård, T., Jensen, R.L. & Maagaard, S.E., 2017. Early Building Design: Informed decision-making by exploring multidimensional design space using sensitivity analysis. *Energy and Buildings*, 142, pp.8–22. Available at: http://dx.doi.org/10.1016/j.enbuild.2017.02.059.

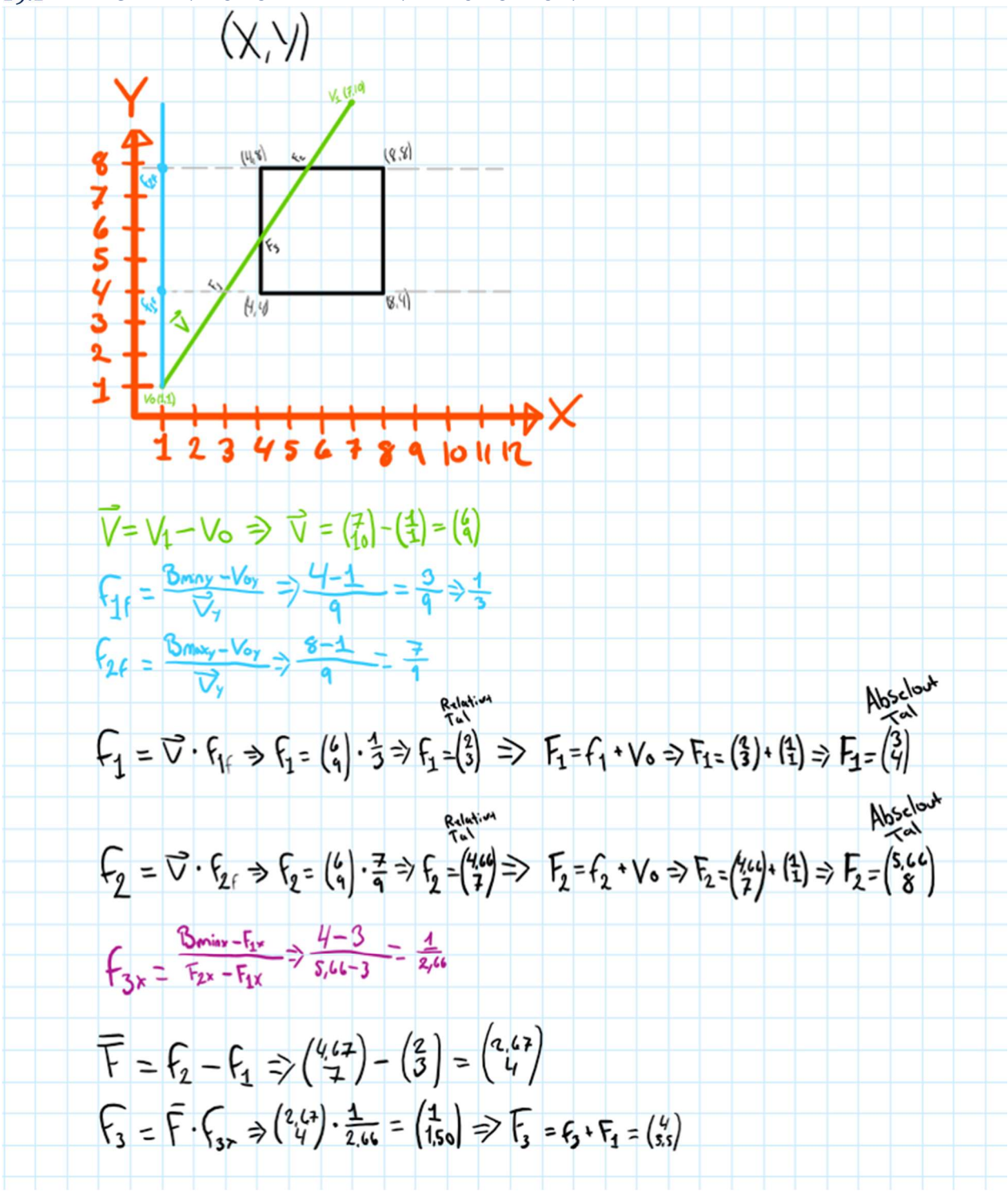
Østergård, T., Maagaard, S.E. & Jensen, R.L., 2015. A stochastic and holistic method to support decision-making in early building design. *Proceedings of Building Simulation*, (Tian 2013), pp.1885–1892.

17 FIGURLISTE

18 TABELLISTE

19 BILLAG

19.1 BILAG 1 – VECTOR AABB INTERSECTION



$\vec{V} = V_1 - V_0 \Rightarrow \vec{V} = \begin{pmatrix}7\\10\end{pmatrix} - \begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}6\\9\end{pmatrix}$

$f_{1f} = \frac{B_{miny} - V_{0y}}{\vec{V}_y} \Rightarrow \frac{4-1}{9} = \frac{3}{9} \Rightarrow \frac{1}{3}$

$f_{2f} = \frac{B_{maxy} - V_{0y}}{\vec{V}_y} \Rightarrow \frac{8-1}{9} = \frac{7}{1}$

$f_1 = \vec{V} \cdot f_{1f} \Rightarrow f_1 = \begin{pmatrix}6\\9\end{pmatrix} \cdot \frac{1}{3} \Rightarrow f_1 = \begin{pmatrix}2\\3\end{pmatrix}$ (Relatief Tal) $\Rightarrow F_1 = f_1 + V_0 \Rightarrow F_1 = \begin{pmatrix}2\\3\end{pmatrix} + \begin{pmatrix}1\\1\end{pmatrix} \Rightarrow F_1 = \begin{pmatrix}3\\4\end{pmatrix}$ (Absoluut Tal)

$f_2 = \vec{V} \cdot f_{2f} \Rightarrow f_2 = \begin{pmatrix}6\\9\end{pmatrix} \cdot \frac{7}{9} \Rightarrow f_2 = \begin{pmatrix}4,66\\7\end{pmatrix}$ (Relatief Tal) $\Rightarrow F_2 = f_2 + V_0 \Rightarrow F_2 = \begin{pmatrix}4,66\\7\end{pmatrix} + \begin{pmatrix}1\\1\end{pmatrix} \Rightarrow F_2 = \begin{pmatrix}5,66\\8\end{pmatrix}$ (Absoluut Tal)

$f_{3x} = \frac{B_{minx} - F_{1x}}{F_{2x} - F_{1x}} \Rightarrow \frac{4-3}{5,66-3} = \frac{1}{2,66}$

$\bar{F} = f_2 - f_1 \Rightarrow \begin{pmatrix}4,67\\7\end{pmatrix} - \begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}2,67\\4\end{pmatrix}$

$F_3 = \bar{F} \cdot f_{3x} \Rightarrow \begin{pmatrix}2,67\\4\end{pmatrix} \cdot \frac{1}{2,66} = \begin{pmatrix}1\\1,50\end{pmatrix} \Rightarrow F_3 = f_3 + F_1 = \begin{pmatrix}4\\5,5\end{pmatrix}$

19.1.1 VECTOR AABB INTERSECTIO – PYTHON KODE DEL 1

```python
import sys

pyt_path = r'C:\Program Files (x86)\IronPython 2.7\Lib'
sys.path.append(pyt_path)

import clr

clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.

def checkTrue(fN, BBminPoint, BBmaxPoint, axisNum):
    output = []
    if axisNum == 0 or axisNum == 1:
        runCounter = 0
        for item in fN:
            if runCounter != axisNum:
                if BBminPoint <= item <= BBmaxPoint:
                    output.append(True)
                else:
                    output.append(False)
                runCounter += 1
            else:
                runCounter += 1
    else:
        for item in fN:
            if item >= BBminPoint and item <= BBmaxPoint:
                output.append(True)
            else:
                output.append(False)
    return output


def PointAxis(axis, point):
    if axis == 0:
        axis = "X"
    elif axis == 1:
        axis = "Y"
    else:
        axis = "Z"
    return getattr(point, axis)


def vectorIntersectionCoordinate(BBpointCoordinate, vector0, vectorV, axisN):
    abseloutVectorCoordinates = []
    for vV, v0C in zip(vectorV, vector0):
        relativeVectorCoordinates = (vV * (PointAxis(axisN, BBpointCoordinate) - vector0[axisN])) / vectorV[axisN]
        abseloutVectorCoordinates.append(relativeVectorCoordinates + v0C)
    return abseloutVectorCoordinates
```

19.1.2 VECTOR AABB INTERSECTIO – PYTHON KODE DEL 2

```
52
53  vectorV = IN[0]
54  vector0 = IN[1]
55  BBminPt = IN[2]
56  BBmaxPt = IN[3]
57
58  intersectionPoints = []
59  doesIntersect = []
60
61  for vec0, vecV in zip( vector0, vectorV):
62      naturalVectorValues = []
63      vectorValuesZero = []
64      vectorCoordinate = 0
65      for vV in vecV:
66          if vV != 0:
67              naturalVectorValues.append(vectorCoordinate)
68          else:
69              vectorValuesZero.append(vectorCoordinate)
70              vectorValuesZeroIndex = vectorCoordinate
71          vectorCoordinate += 1
72
73      intersectPts = []
74      intersection = False
75
76      if len(naturalVectorValues) >= 2:
77          if 0 not in vectorValuesZero:
78              Fx0Coordinate = vectorIntersectionCoordinate(BBminPt, vec0, vecV, 0)
79              if Fx0Coordinate[1] >= PointAxis(1, BBminPt) and Fx0Coordinate[2] >= PointAxis(2, BBminPt) \
80                      and Fx0Coordinate[1] <= PointAxis(1, BBmaxPt) and Fx0Coordinate[2] <= PointAxis(2, BBmaxPt):
81                  intersection = True
82                  intersectPts.append(Point.ByCoordinates(Fx0Coordinate[0], Fx0Coordinate[1], Fx0Coordinate[2]))
83
84              Fx1Coordinate = vectorIntersectionCoordinate(BBmaxPt, vec0, vecV, 0)
85              if Fx1Coordinate[1] >= PointAxis(1, BBminPt) and Fx1Coordinate[2] >= PointAxis(2, BBminPt) \
86                      and Fx1Coordinate[1] <= PointAxis(1, BBmaxPt) and Fx1Coordinate[2] <= PointAxis(2, BBmaxPt):
87                  intersection = True
88                  intersectPts.append(Point.ByCoordinates(Fx1Coordinate[0], Fx1Coordinate[1], Fx1Coordinate[2]))
89
90          if 1 not in vectorValuesZero:
91              Fy0Coordinate = vectorIntersectionCoordinate(BBminPt, vec0, vecV, 1)
92              if Fy0Coordinate[0] >= PointAxis(0, BBminPt) and Fy0Coordinate[2] >= PointAxis(2, BBminPt) \
93                      and Fy0Coordinate[0] <= PointAxis(0, BBmaxPt) and Fy0Coordinate[2] <= PointAxis(2, BBmaxPt):
94                  intersection = True
95                  intersectPts.append(Point.ByCoordinates(Fy0Coordinate[0], Fy0Coordinate[1], Fy0Coordinate[2]))
96
97              Fy1Coordinate = vectorIntersectionCoordinate(BBmaxPt, vec0, vecV, 1)
98              if Fy1Coordinate[0] >= PointAxis(0, BBminPt) and Fy1Coordinate[2] >= PointAxis(2, BBminPt) \
99                      and Fy1Coordinate[0] <= PointAxis(0, BBmaxPt) and Fy1Coordinate[2] <= PointAxis(2, BBmaxPt):
100                 intersection = True
```

19.1.3 VECTOR AABB INTERSECTIO – PYTHON KODE DEL 3

```
                intersectPts.append(Point.ByCoordinates(Fy1Coordinate[0], Fy1Coordinate[1], Fy1Coordinate[2]))

        if 2 not in vectorValuesZero:
            Fz0Coordinate = vectorIntersectionCoordinate(BBminPt, vec0, vecV, 2)
            if Fz0Coordinate[0] >= PointAxis(0, BBminPt) and Fz0Coordinate[1] >= PointAxis(1, BBminPt) \
                    and Fz0Coordinate[2] >= PointAxis(2, BBminPt) and Fz0Coordinate[0] <= PointAxis(0, BBmaxPt) \
                    and Fz0Coordinate[1] <= PointAxis(1, BBmaxPt) and Fz0Coordinate[2] <= PointAxis(2, BBmaxPt):
                intersection = True
                intersectPts.append(Point.ByCoordinates(Fz0Coordinate[0], Fz0Coordinate[1], Fz0Coordinate[2]))

            Fz1Coordinate = vectorIntersectionCoordinate(BBmaxPt, vec0, vecV, 2)
            if Fz1Coordinate[0] >= PointAxis(0, BBminPt) and Fz1Coordinate[1] >= PointAxis(1, BBminPt) \
                    and Fz1Coordinate[2] >= PointAxis(2, BBminPt) and Fz1Coordinate[0] <= PointAxis(0, BBmaxPt) \
                    and Fz1Coordinate[1] <= PointAxis(1, BBmaxPt) and Fz1Coordinate[2] <= PointAxis(2, BBmaxPt):
                intersection = True
                intersectPts.append(Point.ByCoordinates(Fz1Coordinate[0], Fz1Coordinate[1], Fz1Coordinate[2]))


    elif len(naturalVectorValues) == 1:
        if PointAxis(vectorValuesZero[0], BBminPt) <= vec0[vectorValuesZero[0]] <= PointAxis(vectorValuesZero[0], BBmaxPt) \
                and PointAxis(vectorValuesZero[1], BBminPt) <= vec0[vectorValuesZero[1]] <= PointAxis(vectorValuesZero[1], BBmaxPt):
            intersection = True
            if 0 not in vectorValuesZero:
                intersectPts.append(Point.ByCoordinates(PointAxis(0, BBminPt), vec0[1], vec0[2]))
                intersectPts.append(Point.ByCoordinates(PointAxis(0, BBmaxPt), vec0[1], vec0[2]))
            if 1 not in vectorValuesZero:
                intersectPts.append(Point.ByCoordinates(vec0[0], PointAxis(1, BBminPt), vec0[2]))
                intersectPts.append(Point.ByCoordinates(vec0[0], PointAxis(1, BBmaxPt), vec0[2]))
            if 2 not in vectorValuesZero:
                intersectPts.append(Point.ByCoordinates(vec0[0], vec0[1], PointAxis(2, BBminPt)))
                intersectPts.append(Point.ByCoordinates(vec0[0], vec0[1], PointAxis(2, BBmaxPt)))

    intersectionPoints.append(intersectPts)
    doesIntersect.append(intersection)

# Assign your output to the OUT variable.
OUT = intersectionPoints, doesIntersect
```